Connecting Supplier and DoD Blockchains for Transparent Part Tracking

Francis Asuncion, Adam Brinckman, Dwayne Cole, Jeffrey Curtis, Matt Davis, Timothy Dunlevy, Calvin Farmer, Andrew Harrison, Daniel P. Johnson, Joshua Joyce, Chris Klubertanz, Jonathan Lane, John Martin, Jarek Nabrzyski, Joel Neidig, Deysi Olivares, Gregory Robinson, Gabriel Rodriguez, Chris Root, Karen Rowand, Al Salour, Jeff Score, David Scott, Ian Taylor, Chandler Thompson, Huy Truong, Xiqun Wang, Dale Warren

PII: S2096-7209(21)00012-9

DOI: https://doi.org/10.1016/j.bcra.2021.100017

Reference: BCRA 100017

To appear in: *Blockchain: Research and Applications*

Received Date: 1 December 2020

Revised Date: 29 April 2021

Accepted Date: 3 June 2021

Please cite this article as: F. Asuncion, A. Brinckman, D. Cole, J. Curtis, M. Davis, T. Dunlevy, C. Farmer, A. Harrison, D.P. Johnson, J. Joyce, C. Klubertanz, J. Lane, J. Martin, J. Nabrzyski, J. Neidig, D. Olivares, G. Robinson, G. Rodriguez, C. Root, K. Rowand, A. Salour, J. Score, D. Scott, I. Taylor, C. Thompson, H. Truong, X. Wang, D. Warren, Connecting Supplier and DoD Blockchains for Transparent Part Tracking, *Blockchain: Research and Applications*, https://doi.org/10.1016/j.bcra.2021.100017.

Connecting Supplier and DoD Blockchains for Transparent Part Tracking

Francis Asuncion[d], Adam Brinckman[b,c], Dwayne Cole[d], Jeffrey Curtis[b], Matt Davis[a], Timothy Dunlevy[f], Calvin Farmer[d], Andrew Harrison[b], Daniel P. Johnson[a], Joshua Joyce[d], Chris Klubertanz[e], Jonathan Lane[e], John Martin[d], Jarek Nabrzyski[b,c], Joel Neidig[b], Deysi Olivares[d], Gregory Robinson[f], Gabriel Rodriguez[b], Chris Root[b], Karen Rowand[d], Al Salour[f], Jeff Score[e], David Scott[e], Ian Taylor[*,b,c], Chandler Thompson[e], Huy Truong[d], Xiqun Wang[b], Dale Warren[e]

[a]*ITAMCO, Indiana,USA*
[b]*SIMBA Chain, Indiana, USA*
[c]*Center for Research Computing, University of Notre Dame, IN, USA*
[d]*NAVAIR, USA*
[e]*Naval Systems, Inc., USA*
[f]*Boeing Company, USA*

Abstract

Blockchains have been around for more than ten years, and since 2015, a plethora of systems have been launched to target more flexible use cases. More recently, several enterprise blockchain systems, such as Consensys Quorum and Hyperledger Fabric, have been launched to make blockchain simpler to apply in complex organizational configurations. In this paper, we identify a specific Department of Defense use case, extrapolate requirements, and perform a thorough assessment of the different layers of the blockchain stack to identify the existing state of the art and undertake a gap analysis of the technology for this context. We describe a platform that meets many of these challenges and show how we architected, designed, and implemented a solution for this use case for deployment at NAVAIR. This solution connects transactions from two separate blockchain systems, Consensys Quorum and

*Corresponding author.

NAVAIR Public Release 2020-910. Distribution Statement "Approved for public release; distribution is unlimited"

Hyperledger Fabric, by using a graph-based approach that preserves privacy while enabling full transparency across the military and supplier networks.

Keywords: Blockchain, Smart Contracts, Supply Chain, DoD, GraphQL, Enterprise Blockchain systems

1. Introduction

Fundamentally, a blockchain enables a decentralized mechanism for the recording of non-repudiable transactions so that no single entity has control of the data. This is achieved using a distributed consensus algorithm that results in immutable (non-changeable) data, making it impossible for data to be tampered with once written. Ethereum [1] is an example of such a blockchain network that has a network of nodes connected to form an Ethereum Virtual Machine (EVM). On an EVM, transactions are specified through the use of smart contracts, written in Solidity, which define a data and logic interface to the underlying ledger. The public Ethereum network has proved to be scalable and resilient since its launch, having processed more than 800 million transactions [2] at the time of writing.

For enterprise deployments, companies generally use permissioned blockchains, and organizations, such as Enterprise Ethereum Alliance [3] with more than 250 members, help develop new business opportunities and drive industry adoption. Consensys Quorum [4], as one of the more advanced platforms, extends the Ethereum codebase with a set of enhancements to support enterprise needs. Other enterprise systems, such as Hyperledger Fabric [5, 6], aim to provide a modular and extensible open-source system for deploying and operating permissioned blockchains.

Clearly, support for production enterprise blockchain applications is mounting, but still, their application integration is complicated for a number of reasons, including organizational complexity, data integration, sustaining resilience and throughput, version management, authentication, and transaction authorization. In this paper, we provide a real-world application use case from the Verifiability, Identifiability Physical Assets for Real-time Traceability (VIPART) SBIR Phase 2 project that is currently being undertaken at Naval Air Systems Command (NAVAIR). The VIPART project aims to use the SIMBA Chain enterprise platform [7] to track the maintenance, repair, and overhaul (MRO) operations for a tailhook assembly, critical to carrier-based aircraft operations, across the MRO process. This specific use case

reduces the complexity significantly in terms of how one might track every part that NAVAIR deals with, but, at the same time, as a proof-of-concept, it illustrates the complexity of achieving this small slice of the overall picture. The tracking of this single part alone can involve hundreds of operations, tens of data systems, hundreds of data schemas, and multiple data warehouses, which all need to be integrated. After setting the context of this use case, we consider the requirements by dividing a blockchain application into layers to consider each in turn to provide an analysis of existing systems and rationale for choices in this use case. We hope this contribution can serve as a lab book for gaining insights into the architecture of an enterprise blockchain application and help define areas of research that may need to be addressed in the future.

We then provide the rationale for how SIMBA Chain platform was used to architect, design, and implement this use case at NAVAIR. This solution is complex because of the need to meet privacy concerns for NAVAIR data, while offering NAVAIR full transparency for part tracking from OEMs. In other words, OEM data needs to be shared with NAVAIR and NAVAIR data needs to kept private, but NAVAIR would like to query across both its private data and the OEM shared data to have a complete view of the end-to-end process. To achieve this separation, we used two separate blockchains: Hyperledger Fabric for the OEM network and Consensys Quorum for the NAVAIR network. Fabric was selected by Boeing because it was their only supported production blockchain system at the time of writing. We could have used Fabric channels to achieve the separation, but after discussions with NAVAIR, we decided to use a separate blockchain internally at NAVAIR. Quorum was selected for scalability reasons, that are discussed more in Section 4.

To connect the blockchains, we used the platform's ability to create a graph defining relationships between transactions across the two blockchains, which effectively links the smart contracts deployed on the Quorum and Fabric blockchains. This enables complete separation of data to meet privacy concerns while enabling NAVAIR to gain full transparency across their own data as well as their supplier networks. This novel approach essentially creates a searchable graph across both blockchains, which can be queried in order to extract a complete picture for the tracking of an assembly and their parts delivered by external suppliers. While this is a commercial project, we believe this example provides helpful insight into the use of graph techniques for solving practical real-world blockchain application problems, as well as

insight into current gaps where more research is needed, and therefore contributes to the general academic literature and state-of-the-art on this topic.

The next section discusses blockchain-related work in the Department of Defense (DoD). Section 3 describes the DoD use tracking the maintenance, repair, and overhaul process for a Tailhook, along with technical requirements that have an impact on the blockchain architecture. Based on this context, Section 4 provides a walkthrough of a typical blockchain layered stack and addresses the current status. Section 5 describes the architecture we took for this use case. Section 6 discusses the design of the resulting system. Section 7 discusses the implementation of the system. Section 8 provides a retrospective discussion with suggestions for enhancements that would benefit Enterprise deployments. Finally, we conclude the paper in Section 9.

2. Related DoD Work

Maj. Neil Barnas [8] wrote a seminal report entitled "Blockchain in National Defense: Trustworthy Systems in a Trustless World" that researched the possibilities of the use of blockchain technology for the U.S. Air Force. In this work, the application to supply chain was outlined and noted that it could offer a solution that to establish the provenance of every circuit board, processor, and software component from "cradle to cockpit".

In 2018, the National Defense Authorization Act (NDAA) [9] ordered the DoD to "provide to the appropriate committees of Congress a briefing on the cyber applications of blockchain technology" within 6 months. This report included a description of potential offensive and defensive cyber applications of blockchain technology; an assessment of efforts by foreign powers, extremist organizations, and criminal networks to utilize such technologies, an assessment of the use or planned use of such technologies by the Federal Government for critical infrastructure networks; and an assessment of these networks' vulnerabilities to cyber attacks.

In the recent DoD blockchain research study [10], the author used a pharmaceutical industry supply chain use case to determine the feasibility of the approach and its impact on the cybersecurity implications of using blockchain in detecting counterfeit products. This work concluded that it is not yet economically feasible to implement current blockchain technology into the DoD supply chain, primarily due to energy efficiency concerns, data privacy

issues, and the lack of standardized blockchain interfaces among all participants. Recommendations included further advances in the development of blockchain using Proof of Stake (PoS) within Ethereum or Hyperledger to address energy issues and the development of a governance-centered blockchain network that will allow the DoD to act as a regulator, ensuring all actions within the blockchain are monitored and secured. This will address data privacy issues because DoD will determine who can participate, and will standardize all interfaces among participants.

In 2020, a white paper that was supported by Rep. Darren Soto [11] highlighted use cases of blockchain or Distributed Ledger Technology (DLT) that the DoD can benefit from. The report argued that the next generation of emerging blockchain technologies, including Artificial Intelligence, smart drones, robots, and additive manufacturing, will make DoD even more dependent on digital technology and critically dependent on secure, timely, accurate, and trusted data. Yet, as data has grown in importance, cyber warfare has emerged to challenge the U.S. in the digital space. Today, key U.S. defense assets, ranging from communication systems to supply chains, can be disrupted by bad actors attempting to degrade U.S. capabilities.

3. Tailhook Use Case

Maintenance, Repair, Overhaul (MRO) refers to replacements, tests, measurements, and repairs required to keep or restore a component to a usable state. It includes all the actions as well as the procurement of supplies and labor to do so. The MRO category is broad and includes three common types of maintenance that most plants and facilities undertake:

- **Preventive maintenance** are tasks scheduled regularly, designed to keep machines, components, and other equipment functioning well.
- **Corrective maintenance** are tasks performed after a failure occurs and involves fixing or overhauling equipment, machinery, or components, as well as replacing broken components.
- **Predictive maintenance:** are tasks using monitoring and analysis in order to predict future failure incidents and provide remedies.

Here, we consider MRO operations for a Tailhook assembly. A Tailhook is a strong metal bar with a claw-like hook, which is mounted on a swivel on the keel of the aircraft and is normally mechanically and hydraulically held in the stowed/up position. Upon pilot actuation, hydraulic or pneumatic

pressure lowers the hook to the down position, and upon engaging with the arresting gear onboard the aircraft carrier, it achieves rapid deceleration of the aircraft. Figure 1 shows a tailhook being inspected before take off.

Figure 1: Maintenance inspection of an F/A-18 Tailhook prior to launch

This use case aims to track Tailhooks from F/A-18 aircrafts across the MRO process in order to leverage that data for better decision-making. Using a community-oriented blockchain, it is anticipated that such tracking can not only provide a common source of truth for the tracking of such an MRO process but also extend to tracking MRO processes that may be outsourced to Original Equipment Manufacturer (OEM) manufacturers or other qualified MRO facilities or suppliers. Such transparency is currently not available and would lead to significant efficiency gains in terms of coordinating timelines to meet set goals. Another key aspect is the ability to append to the DLT maintenance events and usage information that dictates the ultimate service life of these critical safety parts.

3.1. MRO Flow

To provide context, one needs to have some understanding of the complexity of the Level of Repair Analysis (LORA). LORA is used as an analytical methodology to determine when an item will be replaced, repaired, or discarded based on cost considerations and operational readiness requirements. For complex engineering systems with thousands of assemblies, subassemblies, and components, each having a number of possible repair decisions, LORA seeks to determine an optimal provision of repair and main-

tenance facilities to minimize overall system life-cycle costs. The LORA process starts by identifying options where maintenance can be performed and produces a decision for each item within the system, indicating where each maintenance action for the item will be performed. There are three main levels:

- **Organizational or O-level** maintenance occurs at the organizational unit level, e.g., by a single maintenance squadron as part of an aircraft wing and optimized for quick turn-around to enhance operational availability. O-level is also responsible for the removal of parts.
- **Intermediate or I-level** maintenance occurs in specialized back shops residing at a common operating location. Because it is more specialized, I-level maintenance allows for more thorough and time-consuming diagnostic testing and repair procedures.
- **Depot or D-level** maintenance occurs in highly specialized off-site repair depots or at OEM facilities. They carry extensive diagnostic equipment and possibly even manufacturing capabilities for extensive equipment overhauls and modifications.

The simplified high-level representative MRO flow for a Tailhook, shown in Figure 2, runs from O-Level operations at an aircraft carrier to a depot for repair and back. The procedure runs as follows. O-level removes the part, and if it cannot be repaired by the local I-Level team, it is shipped to a D-level depot for repair. On board the carrier, the I/O level interactions are all recorded on O-level and I-level specific data systems. Once shipped, the part is tracked en route until it arrives at the depot, where it is eventually disassembled and passed to the repair shop. The repair shop interacts with tens of data systems in order to define the work; manage the work orders, time and workload management for staff; and to provide further engineering support. Once the repair is complete, the part will be shipped back to the carrier and installed by the local O-Level operations unit.

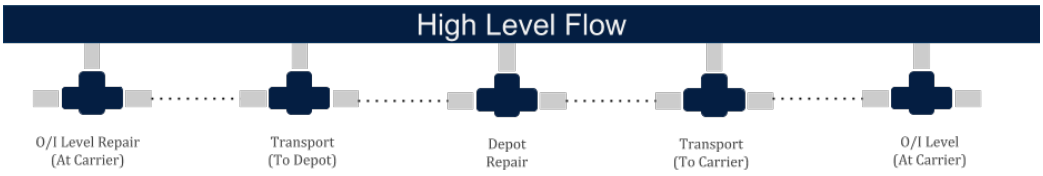

Figure 2: The high level MRO flow for overhauling a tailhook assembly

3.2. OEM Parts Tracking

This use case aims to consolidate tracking information onto a common ledger to track Tailhooks throughout the MRO process. However, often parts are ordered from OEMs during the repair process, e.g., for a Tailhook, this could be a pin, shank, arm, or pivot. Currently, when parts are ordered from an OEM there are no automated mechanisms to track their progress. Some parts are fabricated at the OEMs themselves while others are ordered from sub-tier suppliers, which could order them from further suppliers in the supply chain. Progress updates occur only when parts are delivered to the depot. In this work, we want to assess the possibility of using a DLT to connect information from opt-in OEM suppliers, so that progress updates for certain parts can be tracked in real time. However, to accomplish this, MRO data cannot leave NAVAIR or be shared in any way with the OEMs, while OEM data should be made available to NAVAIR. NAVAIR would like to query across the MRO and OEM tracking data to get a global view of the entire system, which requires two logically separated blockchains in order to separate the data tracking while providing consolidation across the platform.

3.3. Use Case Requirements

As discussed in the introduction, the goal of the use case is to provide OEM (Boeing) part-tracking data to NAVAIR. Trust is key to this scenario because of the two disconnected organizations involved. There are no current means available to share data from OEMs to NAVAIR because NAVAIR systems are completely private and so are the OEMs. To connect these entities it is imperative that the system is secure and private and the data is trusted so that neither party (nor any other adversarial actor) can modify the data on the parts that are being tracked. For example, if an adversarial actor could manipulate the data, then they could disrupt the entire NAVAIR supply chain. Furthermore, NAVAIR wants to be able to query data across their internal (private) systems and across the OEM blockchain part tracking data to monitor their entire end-to-end MRO flow, because the OEM parts being supplied are integral to that process. Therefore, NAVAIR would like to query across both blockchains while OEMs have access only to the data they share.

Certain stages during the OEM assembly process require the operator or inspector to generate a report. The subject of these documents depends on the work done at that stage. For example, at quality assurance work stations, reports may contain the inspector's comments, measurements, the results

of any automated tests, and critical safety tests. A requirement that was identified by the OEM therefore is provide an ability to share such documents and relate them to each part being tracked. Since such documents can be large, the ability to off-chain these documents would be needed.

Consequently, this leads to the following requirements:

- The integration of multiple internal NAVAIR databases for MRO parts tracking.
- Multiple organizations, e.g., NAVAIR, OEM1 participating in a common blockchain network, each having their own governance and security rules, which specify what they share and with whom. Specifically, Boeing would like to share data to NAVAIR in a trusted way and NAVAIR would like access to the entire MRO flow, including OEM data.
- Efficient traceability of the Tailhook process will require structured data that can represent the relationships between the different stages of the process, including tracking the parts of the sub-processes.
- Efficient querying of the resulting data is needed so that data pertinent to a specific part can be queried across the entire MRO process and OEM parts tracking. Also, cross correlations with other parts in order to identify anomalies or to search forward, to identify other parts at risk, are needed.
- The ability to add off-chained data files and associate these with the part-tracking blockchain transactions.
- A Common Access Card (CAC) is a smart card used as the standard identification for DoD employees and eligible contractors and provides access to computer networks and systems. Access to any data or systems at the DoD, therefore, must use the CAC Card authentication.

In the future, this use case will be extended to support the tracking of multiple parts; consequently, smart contracts will need to be updated. It is desirable therefore for the platform to manage smart contract versions so that the application can be updated without incurring significant code rewrites or losing access to data.

4. Analysis of Enterprise Application Requirements

The architecture for a blockchain enterprise application can encompass several levels and components [12]. In this section, we organize our system's

requirements into five sections that are shown in Figure 3. The lower part of this figure focuses on the underlying blockchain systems layers, while the upper part focused on platforms and applications. In this section, we consider each layer, providing a description of the core components that relate to our use case, and identify choices for the underlying blockchain and tools.

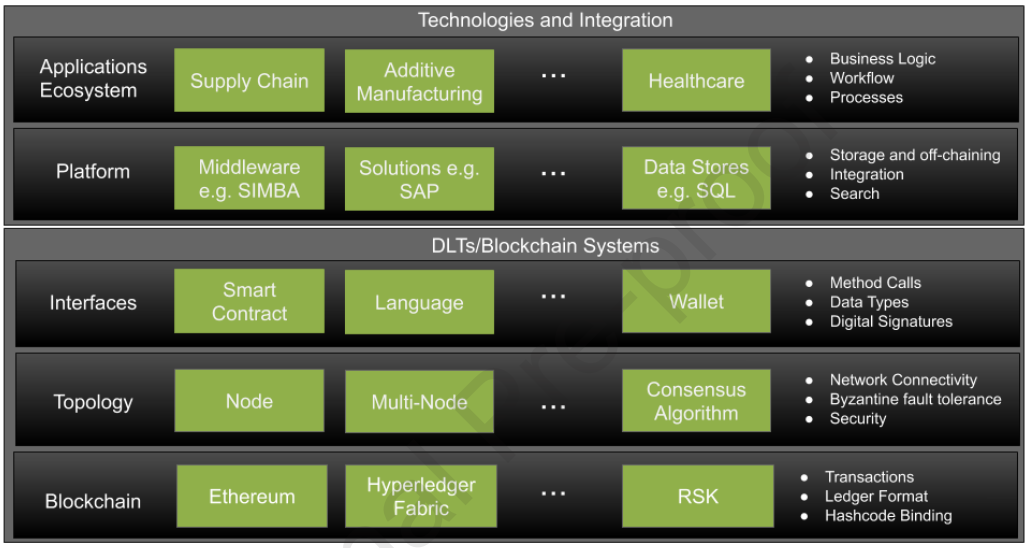

Figure 3: The different layers we analyze for an enterprise blockchain application

4.1. DLT and Blockchain Systems

In this section, we provide a walkthrough of the three layers, namely, blockchains, topology, and interfaces, and finish with a comparative analysis of various blockchain systems using these considerations. The concept of a blockchain has been around in some form or another since the '90s. Stornetta worked with Haber [13] to develop a cryptographically secure archive that could verify records without revealing their contents. Such mechanisms enabled the collaborative creation of digital distributed ledgers with capabilities that far surpass paper-based ledgers. This early work led to the concept of DLT, which can be described as a consensus of replicated, shared, and synchronized digital data geographically spread across multiple sites, countries, or institutions. There is no central administrator or centralized data storage, and as the name suggests, each word contributes to the overall definition:

- Distributed reflects a decentralized rather than centralized nature

- Ledger represents a database of records
- Technology defines the protocol to synchronize data as a decentralized database without a central authority to regulate it.

Consequently, a DLT facilitates the storage of information in a secure and accurate manner using cryptography and, once stored, it becomes immutable. Frequently, DLTs and blockchain are used as synonyms but in fact, a blockchain is a specific type of DLT that encompasses its own set of rules and features, including organizing transactions as a chain of blocks. Blockchains are far more opinionated, defining a specific way of implementing a DLT. This confusion represents a common phenomenon when the success of one type of implementation (blockchain) overtakes the more broad set to which it belongs and becomes more popular than its namesake (DLT).

4.1.1. Blockchain Layer

A blockchain exists on multiple compute nodes at the same time, and data can be written only with consensus from its members; all entries are digitally signed and therefore identifiable; and data can never be removed, i.e., data is immutable. A blockchain constantly grows as new collections of transactions, referred to as blocks, are added to it, using a consensus algorithm. Each block contains a timestamp and a link to the previous block, so they form a chain. The database is not managed by any particular body; instead, everyone in the network gets an exact copy. Old blocks are preserved forever and new blocks are added to the ledger irreversibly, making its data impossible to manipulate. There are three broad types of blockchain networks:

- **Public blockchains** have no access restrictions; anyone can send transactions or participate in the consensus algorithm.
- **Private blockchains** are permissioned, meaning one cannot join it unless invited by the network administrators.
- **Hybrid blockchains** combine public and private blockchains, e.g., an application can run privately and use a public blockchain for settlement.

Some example of public blockchains are Bitcoin [14], Ethereum [15], Stellar [16], RSK [17], and Hedera Hashgraph [18]. In terms of cryptocurrency uptake and to provide an indication of the scale, there are 2728 cryptocurrencies reported [19] at the time of writing. Private blockchain examples are Consensys Quorum [4], Hyperledger Fabric [20], Hyperledger Burrow [21], Ripple [22], and R3 Corda [23].

As discussed in the previous section, NAVAIR requires a private blockchain network and the organizations where the blockchain nodes are owned and operated form part of the overall security model. Reports indicate that there are about 15 enterprise blockchain systems, but after a recalibration of expectation following a Gartner's infamous trough of disillusionment[24], there are three quite separate enterprise-focused blockchain networks at the core of today's technology: R3 Corda, Hyperledger Fabric, and enterprise Ethereum. Furthermore, several sources, e.g., [25] and [26], consistently list Fabric, Corda, and Quorum as the top three enterprise private blockchains at the time of writing. For this use case, therefore, we considered those three systems. Enterprise applications require administrative control to grant user or company participation; consequently, permissioned blockchains are typically used. For data confidentiality, Corda, Hyperledger Fabric and Quorum all address transaction security but handle it differently.

Corda does not require all nodes in the network to run all transactions. As a result, the whole blockchain state can be determined only through a union of all nodes in it. Transaction visibility is based on a need-to-know basis, which means all transactions need be visible only to those involved in them. This inherently provides visibility scoping. Additionally, it improves throughput, because not every node on the network needs to compute all transactions.

Quorum provides a mechanism to encrypt transaction data and store only the hash of the transaction information on the blockchain. This means the transaction is visible to the whole blockchain but only in encrypted form. An extension to the Ethereum protocol allows nodes to share unencrypted transaction data with a subset of nodes on the network, providing simple and dynamically defined visibility scoping based on use case.

Fabric uses the concept of "channels" through which agreed participants exchange data. Channels are essentially subnets within the overall blockchain network. Trusted partners share channels, and data is visible to only those partners who have access to the channel. Channels are statically configured between agreeing parties. The configuration and maintenance of channels can add complexity to a deployment if there are many partners involved with overlapping security and authorization requirements. Swift Bank's Nostro POC [27], which used Fabric stated, "For example, while 528 channels were required in the PoC to ensure Nostro accounts would only be stored on the nodes of their account servicers and owners, to productize the solution, more

than 100,000 channels would need to be defined, covering all existing Nostro relationships, presenting significant operational challenges."

As stated in the introduction, Fabric was selected for the OEM network because it was the production blockchain system Boeing uses. For the internal DoD network, however, we did not select Fabric because of the aforementioned scalability issues for scaling out Fabric across multiple DoD stakeholders of the MRO pipeline. The selection between Corda and Quorum was driven by the smart contract language at the time of implementation. SIMBA Chain supports any Ethereum-based blockchain and it also supports Fabric. However, Corda uses Kotlin or Java for their smart contract language; this would have resulted in further integration, which was out of scope for this project.

4.1.2. Topology

Topologies differ vastly between public blockchains and private blockchains. Public blockchains are more decentralized, have more participants with no trust, and use compute intensive consensus algorithms. For example, Bitcoin has about 9,000 computer nodes participating in its "proof of work" consensus algorithm with at least 51% agreement needed, which is costly and slow. A private blockchain has members that are more trusted and have multiple enterprise-oriented advantages including: simpler consensus algorithms with far superior performance; low latency rates; users or companies are granted access rights and roles, often centrally administered, providing the necessary control on who can participate and how; and flexible models for the on-chain security of data.

The concept of "consensus" is the process of achieving a non-disputable agreement in the resulting states among the participating nodes, for the addition of a new block to the chain. The increase of demand in enterprise blockchains has led to more efficient consensus algorithms, with differing approaches competing in this area.

The Corda consensus is at the transaction level and supports two types. Validity consensus checks that the transaction is accepted by the contracts of every input and output state and has all the required signatures. Uniqueness consensus is the requirement that none of the inputs to a proposed transaction have already been consumed in another transaction.

Fabric uses orderer nodes to establish the order of all transactions, then batches of transactions are arranged into blocks that are later added to the blockchain. Orderer nodes broadcast blocks to connected peers, which relay

to other peers via gossip protocol. Peers then validate broadcasted blocks from the orderer node and then submit them to the ledger. There are three interchangeable consensus protocols in Hyperledger Fabric Solo, Raft, and Kafka - that can be used.

Quorum uses a Proof-of-Authority (PoA) reputation-based consensus algorithm that leverages the value of identity and reputation of block validators. PoA networks designate certain nodes that are trusted, i.e., have authority to validate transactions. The BFT consensus algorithms used (e.g. Probabilistic BFT and Istanbul BFT) provide transaction finality rather than probabilistic completion, which is the case in PoW. BFT combined with PoA means it is possible to ensure there can be no forking of the chain. This, in turn, equates to finality, and because finality is guaranteed, there is no need to wait for block confirmations to reach a level of completion probability that participants are comfortable with. Therefore, again, throughput is increased.

4.1.3. Interfaces

In this layer, we discuss two items: smart contracts, which provide an interface to the blockchain, and wallets, which act as an interface to a user. Both elements differentiate blockchain from most other storage systems. Smart contracts provide the data interface to what is written on the blockchain and the business logic of what rules need to be satisfied for this 'write' operation to take place. A smart contract is a self-executing piece of code that defines the terms of the agreement for exchanges, or state changes for an asset. Smart contracts are often written into the blockchain, making them non repudiable, to offer deterministic behavior and are trusted, trackable, and irreversible.

Unlike other data storage systems, each blockchain transaction is digitally signed using cryptographic signatures, stored in a Wallet. The term wallet originated from public networks where the special private key provided access to your cryptocurrency. However, really, a wallet is a cryptographic credential that identifies a user on the network. It is an asymmetric cryptosystem (public/private key pair) that is used to sign transactions, making each transaction identifiable, where the private key is kept secret, the public key can be made public and the wallet address provides an address on the network, typically generated by taking a hash of the public key.

4.2. Technologies and Integration

This section provides some insights from different platforms and applications, along with some of the issues that need to be managed.

4.2.1. Platform

Enterprises should not interact directly with a blockchain using the low level blockchain APIs. Rather, they need an application integration layer that sits above the blockchain, abstracting blockchain-specific details and allowing integration of other enterprise systems.

Beyond the features a blockchain network can provide, there are capabilities that an Enterprise Blockchain Platform (EBP) should provide to allow enterprises to seamlessly execute their business processes in a scalable, secure, and robust way. We elucidate what such a platform should provide below.

Scalable and Robust Transaction Execution. Along with ease of integration, transaction execution should be scalable and robust. This means submissions should employ techniques supporting retries and idempotency, asynchronicity, and horizontal scaling. Asynchronicity allows transactions to be queued on the application side if throughput is in danger of overwhelming the blockchain network. Horizontal scaling means leveraging the decentralized nature of the blockchain to push transactions to different nodes on the blockchain, meaning submission to the network does not become a bottleneck to throughput.

Enterprise Auth Mapping. Enterprises have complex and highly integrated security systems, typically defining rules for authentication and authorization using centralized role-based backends, such as LDAP or Active Directory. An EBP should provide mechanisms to map the authentication and authorization rules to the identities on the blockchain the platform provides access to. This means mapping users and/or roles to transaction-signing identities, as well as integrating on-chain smart contract execution control to the roles defined within the enterprise.

Event-Driven Models. Many enterprise systems are event-driven and many enterprise blockchain use cases are intrinsically linked to the concept of events occurring and being recorded. As a result, it is vital that an EBP provides mechanisms to allow systems to subscribe to and be notified of blockchain events in order to trigger higher-level actions, reactions, and workflows in the

system. Notifications should be possible both to human users and to other enterprise sub-systems and external consortium partners. As an example, NAVAIR would like to get notified upon blockchain updates for particular parts as soon as their status changed. This would enable them to perform demand sensing and re-plan their logistics.

External Data Integration. The storage of large data on a blockchain is not recommended because it is expensive and inefficient to scale, so often *off-chaining* of data is used. Off-chaining allows the data to be stored in an external data store but bound on the blockchain by storing its fingerprint or hashcode on chain. A hashcode is a cryptographic mechanism that can generate a small, fixed-sized fingerprint for any size file, in such a way that it is virtually guaranteed two files will never generate the same fingerprint. This means that a hashcode uniquely represents the content of a file. Consequently, if we store the hashcode of a file on the blockchain, and the file in another store, we guarantee that: the contents of the file have not been changed since it was stored because the hashcode is unique to that content; and the user that stored the external file is identified because each transaction is signed by the user, using their wallet.

Off-chaining is an excellent strategy for tracking external files or datasets on the blockchain because it minimizes the storage on-chain while providing a non repudiable audit trail for externally generated data. This means it is possible to seamlessly use blockchain with existing data and systems, to vastly improve the tracking of data, the integrity of data, and aggregating multiple copies of data across disparate data systems and applications.

For our use case requirement, most of the OEM documents are expected to contain sensitive information. The hash of the document can be recorded on the chain, along with information associating the document with the part, assembly stage, and other data pertinent to tracing the document. The hash recorded on chain provides an immutable and irrepudiable way of proving that the document was never altered after the status of work was written to the ledger. This is possible because the blockchain consensus algorithm ensures that all peers have an uncorrupted copy of the ledger. Consequently, off-chaining is a necessary component of the resulting system.

Blockchain Search and Querying. In order to perform complex searching, relationships between transactions need to be identified in order for queries to be extracted. For cryptocurrencies, relationships between wallets that have

transacted with each other may be insightful. For example [28] describe how the semantics of blockchain transactions can be captured by a directed acyclic graph, representing states and transactions between the states, together with conditions for the consistency and validity of transactions. [29] explains how graph mining has become a major part blockchain analysis and provides an overview of such graph techniques.

However, for more general enterprise smart contract, the methods and data are custom to a specific application. For our use case, we would like to query for a specific part and other transactions that relate to that part. In the Web3 community, initiatives such as "The Graph" [30], are emerging that provide a mechanisms for indexing and querying data from blockchains. It makes it possible to query data that is difficult to query directly by enabling curators to create "subgraphs", which define the data "The Graph" will index. In other words, curators take time to understand the transactions and how they relate to each other in order to create a GraphQL schema and mapping that others can use. Since it now supports several blockchains, it technically could form a mechanism to create a global graph of blockchain data to represent the two blockchains in the use case in this paper. Anyblock [31] is another tool that employs the use of Elasticsearch and PostgreSQL to support to search, filter and aggregate across multiple Ethereum based blockchains. However, the data would need to be curated and tied together for this to work, so domain knowledge would be needed.

4.2.2. Applications Ecosystem

Successfully implementing a blockchain-based solution requires collaborating with partner and competitor organizations, making blockchain a "team sport," and for the DoD this often means a "consortium" of companies and government. Data sharing across what have historically been tight corporate boundaries can only change in circumstances where there is a clear benefit for all. Negotiating such boundaries and rules about what data may cross boundaries and under what conditions is a significant part of the solution. Indeed, for our use case, sharing rules are specific and for both OEMs and the DoD, with such data sharing being an unfamiliar concept.

Most businesses use Enterprise Resource Planning (ERP) software to incorporate the key business functions of an organization and offer query components that can provide customizable access to data, including lists, channels, and reports. Interfacing ERP systems with blockchain is complex but essential if one is to remove the human in the loop and introspect the

multiple levels of supply chains automatically.

At the application levels, EBP should also provide ways for an enterprise to truly integrate blockchain into their business processes. While the above capabilities enable utilization of blockchain, they are only the first step towards driving business processes through smart contract transaction execution. To achieve this, enterprises should be able to do two things:

1. Encode business process rules into smart contracts. A simple way to achieve this is to consider the smart contract as a finite state machine (FSM) whose states and guards are expressions of business process requirements. By encoding these rules into the contracts, failure to execute or transition states when a contract method is invoked can trigger downstream business process events.
2. The EBP should provide mechanisms for modeling business-level transactions that might be made up of multiple blockchain transactions executed by different parties. There are various approaches to this problem. While rollback is not a possibility on a blockchain, such business level transactions can be modeled using the Unit Of Work (UOW) pattern [32]. Data can be aggregated during a business-process transaction and finally written once to the blockchain, or transactions can be written individually and reference a UOW identifier capturing each event as well as possible failure.

5. Architecture

Using the use case requirements in Section 3.3 and the analysis performed in the last section, we identify here the components required to meet the demands of this use case and how they interact. Figure 4 shows the resulting architecture for the NAVAIR OEM use case.

In this figure, the information flows from top to bottom. To simplify, we've separated the platform components from the other components in the system. Information from NAVAIR and OEMs flow from their database and ERP systems that contain the raw data for internal tracking purposes. This information is collected using different mechanisms for OEMs and NAVAIR, and details are out of the scope of this paper, but the result is that data is aggregated into a form for consumption. In this case, we required the tracking information for the Tailhook assembly and any corresponding parts that are ordered during the MRO process. For the OEM network, there are currently two OEMs, Boeing and ITAMCO, who each have their own

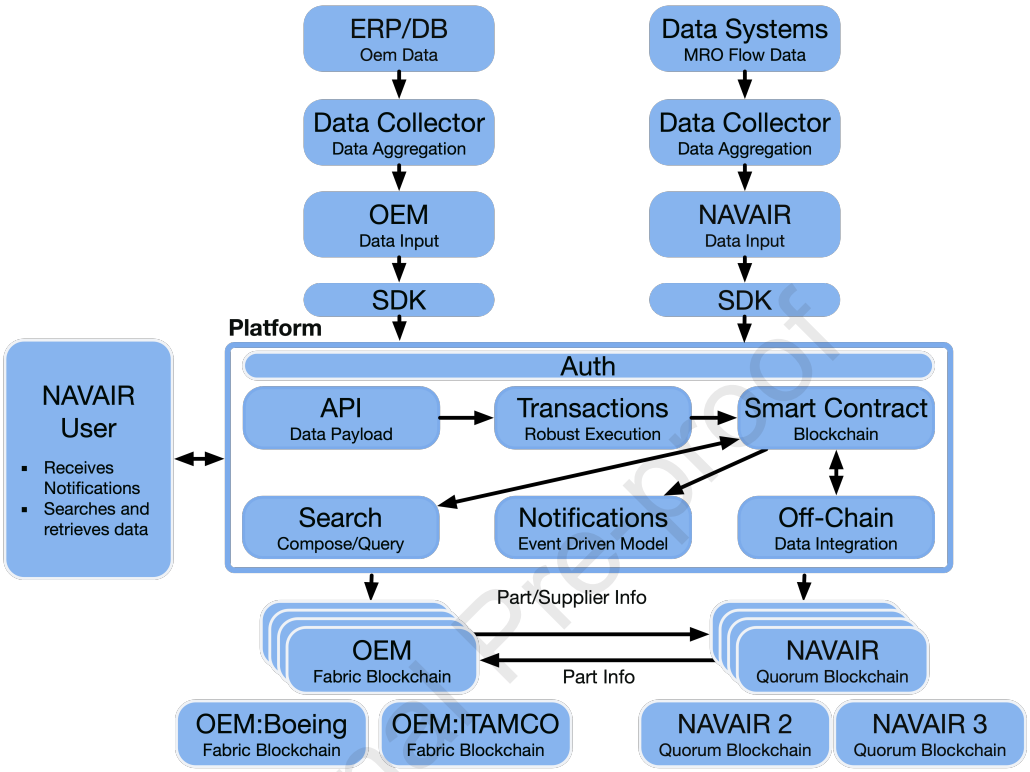

Figure 4: Architecture for Addressing Use Case Requirements

blockchain node. Currently, the NAVAIR nodes are in one administrative domain but in future the idea is that there will be nodes at each part of the MRO pipeline and owned by different NAVAIR units.

For convenience, the input data from either party is input into the blockchain platform using a Software Development Kit (SDK). The SDK fills the gap between the raw data and the data format that is expected by the smart contract interface of the platform. The SDK also makes the authentication and authorization simpler by providing boilerplate code for integration. The SDK will communicate with a platform API, which abstracts the details of the blockchain infrastructure from the application, making it portable, as discussed in the previous section. The platform provides the enterprise auth infrastructure necessary to host NAVAIR and the OEMs. At NAVAIR, the authentication scheme needs to use CAC cards, and for OEMs, support for the typical OEM auth should be provided.

At the platform level, the features that were identified in Section 4.2.1

should be supported to provide the core resilience necessary for effective tracking. The complexities of the blockchain should be abstracted using an API that supports the necessary data schema, then, on the backend, robust monitoring and asynchronous delivery of transactions should be supported to help mitigate against peak data loads and provide resilience for transaction throughput. Transactions should be interfaced with the specific blockchain, and off-chaining of data should be provided to support any larger files or data segments that accompany on-chain transactions. Then, for interfacing with the users, two sub-systems should be provided to: provide notifications for certain transactions if they meet some criteria; and search facilities that can effectively search the blockchain data in a fast and efficient way to support the end users in tracking the specific parts and help identify issues. Finally, the smart contract language and blockchain system that each network would like to use should also be integrated to meet the use case demands.

6. Design

The goal of this use case is to provide OEM part-tracking data to NAVAIR, who want to be able to add this tracking data to their internal data so that they can query and monitor their entire end-to-end MRO flow. In order to achieve this vision, we need to construct two blockchain permissioned networks to enable the data tracking across two disconnected organizations. We then need a system that can connect these blockchain systems together for querying purposes. It is imperative that the blockchain systems are secure, private and the data is trusted so that no party, or any other adversarial actor, can modify the data about the tracked parts. For the NAVAIR installation, the system also needs to provide CAC card access. The system also needs to be capable of scaling out as transaction throughput is increased, it needs to supporting events, efficient searching and meet the other technical requirements outlined in Section 4.

There are a number of toolkits that are used by developers in order to develop blockchain applications. Truffle [33] is a development environment, testing framework and asset pipeline for blockchains using the EVM to make development simpler. Truffle is designed for blockchain developers, not for enterprise developers, but is an incredible platform for built-in smart contract compilation, linking, deployment, binary management, and automated contract testing. It has an extensible deployment and migrations framework.

Truffle is the de facto toolkit that interfaces with the EVM, and almost all higher level tools utilize it.

Remix IDE [34] is an open source, browser-based compiler where users can write Solidity smart contracts. Developers can use Remix in the browser as well as locally, and it supports testing, debugging, and deploying of smart contracts. Kaleido [35] streamlines the process of standing up secure, permissioned blockchain networks without sacrificing the ability to customize the environment. These networks offer all of the benefits of the underlying blockchain technology, while still maintaining the necessary levels of robustness, security, and performance. The platform onboards member organizations and distributes ownership and control among them according to defined and agreed governing policies. Chainlink [36] enables smart contracts to access any external data source by using a decentralized oracle network, e.g., sports results, the latest weather, or any other publicly available data.

MultiBaas [37] is a platform that enables the development of Ethereum blockchain applications using a familiar REST API. MultiBaas enables role-based access control, two-factor authentication, and audit logs that keep things secure. Settlemint [38] enables developers to build blockchain applications in a browser using the Visual Studio Code IDE but provides a variety of smart contract templates for various use cases to bootstrap development. It also generates a REST API to interact with smart contracts and provides management tools for adding users and connecting to infrastructure.

These aforementioned platforms, however, do not support transaction resilience or multi-chain scenarios. The two systems that come the closest to meeting the requirements are Codefi Orchestrate [39] and SIMBA Chain [7]. Codefi Orchestrate provides developers with enterprise-optimized features on top of the ConsenSys Quorum blockchain and the Hyperledger Besu client [40]. Codefi Orchestrate key features include: transaction management and resilience, account management, smart contract registry, chain management, high availability, and multi-tenancy. Orchestrate also has functionality for managing multiple tenants and multiple blockchains simultaneously. For the lower levels of the stack, Codefi Orchestrate satisfies the majority of the requirements. However, there are other requirements it does not satisfy. First, although it supports multiple chains independently, it does not provide ways of connecting transactions from different blockchains together in a single application. Secondly, it does not offer search mechanisms that can retrieve application data that may include different smart contract versions and blockchains.

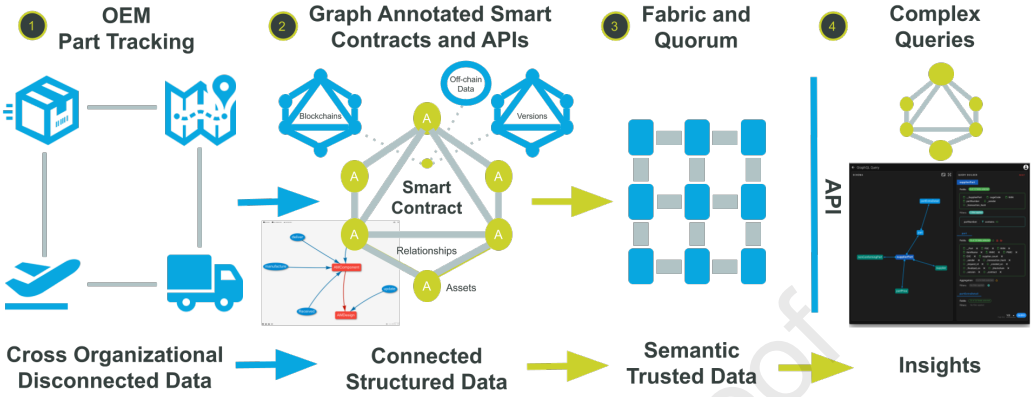

Figure 5: An flow of the Data and Relationships preserved for the SIMBA Chain VIPART Application

To be able to query along the part tracking process, a flexible query mechanism capable of extracting relationships between transactions needs to be realized. A user should be able to search across the entire application, even though the data for that application may be collected using different smart contracts, different smart contract versions, and even on different blockchains. NAVAIR users need a single unified view of the entire process. This is where the SIMBA Chain enterprise platform excels over and above the systems. The SIMBA platform is a comprehensive, fault tolerant, secure environment that is focused at providing scalable production enterprise applications. An underlying Graph-based model connects everything together by representing relationships between entities, connecting versions of smart contracts, and even mapping smart contracts between different blockchains for a single application.

Figure 5 shows how one would use SIMBA Chain to track parts using the SIMBA chain approach. Data from the supply chain interfaces through smart contracts before being stored on the underlying blockchain. SIMBA Chain applies the use of the graph model to annotate smart contracts with data relationships between the tracked assets and the methods whereby those assets are being tracked. Essentially, this model conceptualizes an application's data and relationships using assets to define the nouns of a business process, and transactions to describe the verbs, or relationships. It can be specified by using SIMBA's Smart Contract designer GUI, as indicated, to create the structure and generate the smart contract code from scratch or it can be annotated to existing smart contracts. Once applied, the on-chain

transactions are aware of what they are connected to and collectively form a sophisticated graph of the entire application's data. This enables complex queries on the right of Figure 5 to be made using GraphQL [41] across the application, including the ability to link all smart contract versions and from transactions on one blockchain to another. A screen shot of the GraphQL tool is given to illustrate this concept. SIMBA Chain can search the entire graph; meaning that a single search can traverse the same application that could be co-hosted on several blockchains and contain several different versions. Using this approach, future-proof multichain applications can be built because the graph will continue to extend as the production application evolves to provide a single unified view of the entire application. Figure 6 provides a full list of SIMBA Chain features that can be applied for this use case.

	Feature and Purpose	Benefit	Benefit
1	**SIMBA Chain Model** Common graph format for representing relationships in blockchain applications	**Semantic Relationships** Provides a semantic overlay that defines how transactions stored on the blockchain relate to each other	**Linked Transactions** Link transactions, smart contracts methods, smart contract versions and transactions from one blockchain to another
2	**Smart Contract Designer** Provides a low-code/no code GUI to specify an application using the SIMBA Chain model	**Simple Smart Contracts** Makes it simple to graphically build relationships (assets and transactions) to rapidly design smart contracts	**Extendable** SCD server component is modular and can be easily upgraded to support new solidity versions
3	**Virtual Application APIs** Smart contract methods exposed as REST endpoints for simple development and Enterprise integration	**Well Known Developer API** Simple, well known, REST interface help developers integrate blockchain with Enterprise applications	**Custom and Documented** Custom application API, specific for business processes and payloads. Full API documentation using Swagger
4	**Transaction Validation & Resilience** Transaction validation, nonce management, load balancing and resilience	**Validation And Nonce Management** Validation ensures payload is valid. Transaction nonce is manage for high throughput	**Asynchronous Transactions** Each transaction is spawned off into a scalable queue to monitor completion of every transaction
5	**Blockchains and Multichain** Support for multiple blockchain bindings and multiple blockchains per application	**Multiple Blockchain** Ethereum, Consensys Quorum, Hyperledger Fabric, Binance, Burrow, RSK, Stellar	**Multichain** Graph relationships connect assets on one chain with another blockchain to support multichain scenarios
6	**Transaction Cache** Cache blockchain transactions in a DB to support fast searching capabilities	**Transaction Monitoring** Monitors transaction state, completed transactions are cached to PostGres: metadata + JSON payloads	**Fast Searching** Cache makes searching blockchain data simple and lightning fast
7	**Search And Access APIs** Developer APIs for searching blockchain transactions (cached or on-chain)	**Search Filtering** Search and filter blockchain data per method or per smart contract by parameters	**Graph Searching** GraphQL querying of stored relationships between transactions, contracts and blockchains
8	**Off Chain Data** Support off-chain data associated with a on-chain transaction, including single files and multiple files	**Multiple File Storage Support** Data stores supported: IPFS, Ceph, Azure Blob storage	**Bundling Files** Data bundles create a manifest that stores hashcodes to multiple data files using content-based hashes
9	**Transaction Subscriptions** To be able to subscribe to blockchain transactions. Filter subscriptions by parameters and Support multiple targets	**Logic Based Subscriptions** Filters can be chained to create logic expressions to notify on specific situations e.g. notify if {id=8756} & {name=Ian}	**Multiple Targets** Support SMS, Email and Web notification targets; configurable to provide authentication e.g. SMS, SMTP
10	**Dashboard** Provides a Web interface to functionality, smart contracts management, subscriptions and search	**Smart Contracts** Smart Contract Designer allows zero-code contracts to be written, which can be deployed onto any blockchain	**Management and Querying** Subscriptions, orgs, users, authorization can be managed. Interactive and visualized blockchain transaction queries
11	**Auth** Authentication and Authorization bindings to existing Auth infrastructure for seamless Enterprise integration	**Authentication** Enterprise identity integration examples: Azure Active Directory and DoD Common Access Card (CAC)	**Authorization and Identities** Smart contract permissions and wallet auto-generation for identities using secure key stores in multiple environments
12	**SDKs** To make client development simple and to streamline client-side transaction signing	**Languages Supported** We maintain SDKs in Python, Java, Node and .NET	**Environments Supported** Support for the public and enterprise environments

Figure 6: SIMBA Chain Enterprise Environment Features

The main feature the Department of Defense find important is to have a single version of the truth of data between organizations. One challenge is ensuring that all parties from the Aircraft Squadron, Military Depot, purchasing, and Tier 1 OEM suppliers agree on the same part information, tracking,

and related dates. SIMBA Chain's studies have found discrepancies between systems that rely upon manual data hygiene techniques, which are often independent and blind to the other organizations. Additionally, within the military, the concept of Commander's Critical Information Requirements is a formal process and culture that revolves around the decision-making a commander makes based upon the prioritization of critical information, timely and accurate intelligence reports. The key feature for the military is the audibility and record-keeping of the provenance of data and the time. More importantly, when inferences are made and command decisions executed, The military has a transparent graph and trustworthy record. The solution provides a graph of queried datasets backed by the immutability of the blockchain. In short, when command decisions are executed and then later reviewed retrospectively, we may begin to answer the question, "what did we know, and when did we know it?". Other important aspects are to be able to extend the system when new data is needing to be tracked. Providing a simple way to achieve that will enable NAVAIR to extend the system, create new smart contracts and tie everything together seamlessly.

SIMBA's 12 components shown in Figure 6 are designed to satisfy theses main criteria. SIMBA is designed to make it easy to develop and extend blockchain applications and interface with them using GUIs for creating smart contracts and for auto-generating application-focused APIs (see Section 7). SIMBA also provides a state-of-the-art robust, fault tolerant and scalable blockchain platform that preserves blockchain or data store independence and is designed ground up for production enterprise applications. The platform provides full integration with existing authentication/authorization frameworks, including CAC card integration for meeting the DoD use case security requirements. It also supports asynchronous transaction processing and monitoring, which manages cryptographic nonces, in order to load balance and ensure transactions are successful. It also supports transaction validation to check whether the parameters conform to the specification to return transactions that do not conform.

In terms of sustainability, SIMBA's platform supports multiple blockchains and provides seamless integration of off-chain data to many stores, including Ethereum, Consensys Quorum, Hyperledger Fabric, Binance, Burrow, RSK, and Stellar blockchains and IPFS, Ceph, and Azure Blob storage data stores. Graph relationships can be made from an asset or transaction on one chain with those on another blockchain to support multichain scenarios bringing all of the data together in one place including

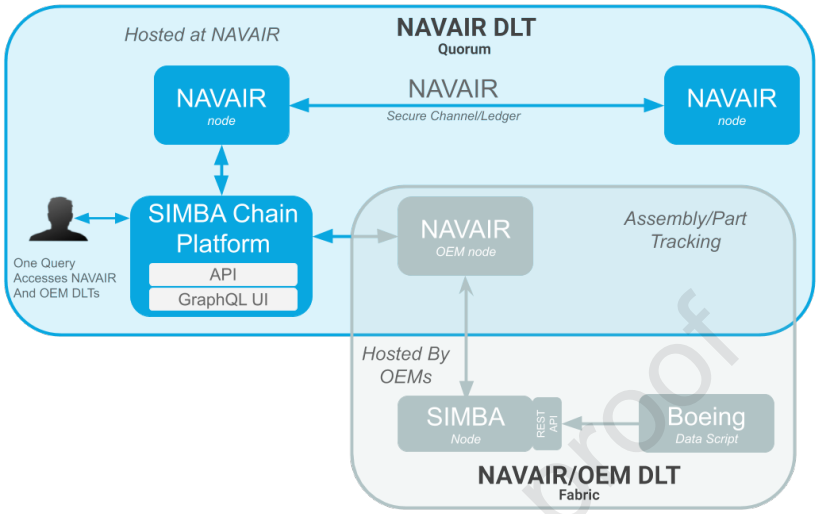

Figure 7: The consortium blockchain architecture

smart contract versions. For the user, SIMBA provides filtered searches at the method level, or smart contract level, and a GraphQL search API, which allows a developer to search an application's data relationships. Finally, users can subscribe to blockchain transactions and events and filter subscriptions by using logic expressions.

7. Implementation

The SIMBA Chain platform was used to implement the pilot at NAVAIR. The configuration for the blockchain is shown in Figure 7. This configuration provides connectivity of the data from our OEM partner for the pilot, Boeing, through a proxy hosted on the SIMBA Chain node of the OEM DLT, as shown. The preference from the OEM was to use Hyperledger Fabric and for internal NAVAIR use, we selected Consensys Quorum to simplify scaling to the multiple data systems in the MRO process. The resulting configuration places a SIMBA Chain platform at NAVAIR, which has access to the data from both Quorum and Fabric blockchains.

The OEM Fabric network track parts that OEMs might supply and the Quorum ledger tracks internal NAVAIR MRO flows. In the case of Boeing, we are also able to track progress down to its sub-tier supplier (tier 2) to gain finer levels of granularity of tracking. Application data is fully available at NAVAIR for querying across the ledger, using the SIMBA Chain GraphQL.

To achieve Fabric integration, we needed to integrate Hyperledger Burrow [21], which is described in the next section.

7.1. Hyperledger Fabric Integration

Hyperledger Fabric DLT is used for OEM transactions and the veracity, or the truth, of the ledger will be maintained by a consensus among all Fabric peers on the network. This network is a private network, and all communications between peers are tightly secured by a Hyperledger Fabric membership service provider (MSP) and mutual TLS.

Since Fabric by itself does not support Solidity smart contracts, we integrate Burrow EVM chaincode onto peers to act as a shim between EVM-based adapters (implemented by SIMBA) and the Fabric peer. To further abstract the EVM chaincode, we also deploy a proxy that translates EVM Remote Procedure Call (RPC) protocol requests into Fabric chaincode instructions. This enabled us to reuse all of the tooling to include Smart Contract design and compilation, caching of the ledger data for faster querying, and off-chaining large datasets in a non-repudiable way into distributed file systems.

Briefly, this was accomplished as follows. The Fabric Peer has access to decentralized ledger data. On the ledger, we have something called a chaincode namespace, which we called 'EVMCC.' Under this namespace, all smart contracts and their data can be found. When a call is made from one of our EVM adapters, the Burrow EVM chaincode looks for this namespace on the ledger. If it has been found, it looks up the contract within this namespace using its address. The address, in this case, is semantic since Fabric has no notion of contract addresses. Burrow EVM then loads the contract bytecode into memory and processes the code as it would be executed on an ordinary Ethereum blockchain. The only difference here is that the contract code is executed by a subprocess of the chaincode installed on the peer. During this integration, we fixed two bugs and pushed those back into the Burrow Fabric repository for other community members to use.

7.2. The SIMBA Chain Proxy

The SIMBA Chain node consists of a proxy API interface that enables Boeing to pass CSV files, containing the supplier information and status, for tracking the parts supplied by Boeing. Once collected, the data is then interfaced with the SIMBA platform to transact the information on the blockchain. Essentially this process proceeds as follows:

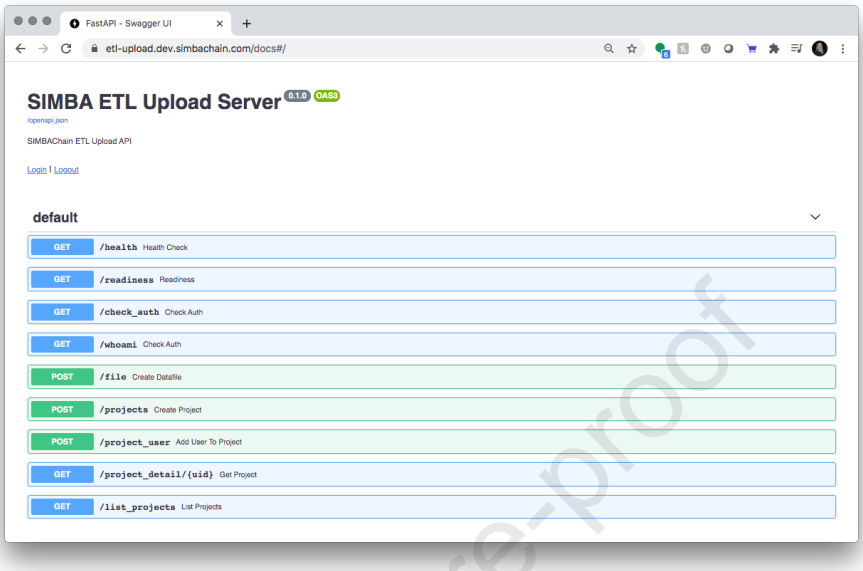

Figure 8: The ETL Swagger interface to support OEM

- Boeing queries their ERP systems to extract information for the parts we are tracking periodically, using Apache Airflow.
- Boeing authenticates to the SIMBA Chain Extract, Transform and Load (ETL) API using Oauth and posts the CSV files to the endpoint.
- SIMBA Chain ETL parses the data and translates it into a format that can be transacted using SIMBA's API on the secured shared DLT

The ETL API interface is documented using Swagger[42], as shown in Figure 8. In the POST payload, the OEM ID is provided and the CSV file is attached using a multipart form POST. In the future, as the OEM network grows, each OEM could use the SIMBA Proxy approach to connect (and not have their own node) or they could stand up a Fabric node and connect directly.

7.3. Smart Contract Development

Two smart contracts have been developed for this use case, one for the NAVAIR Quorum network and the other for the Fabric OEM network. To develop these contracts, we took three steps:

- To specify business rules, that is, to specify which assets to track and how to track them.

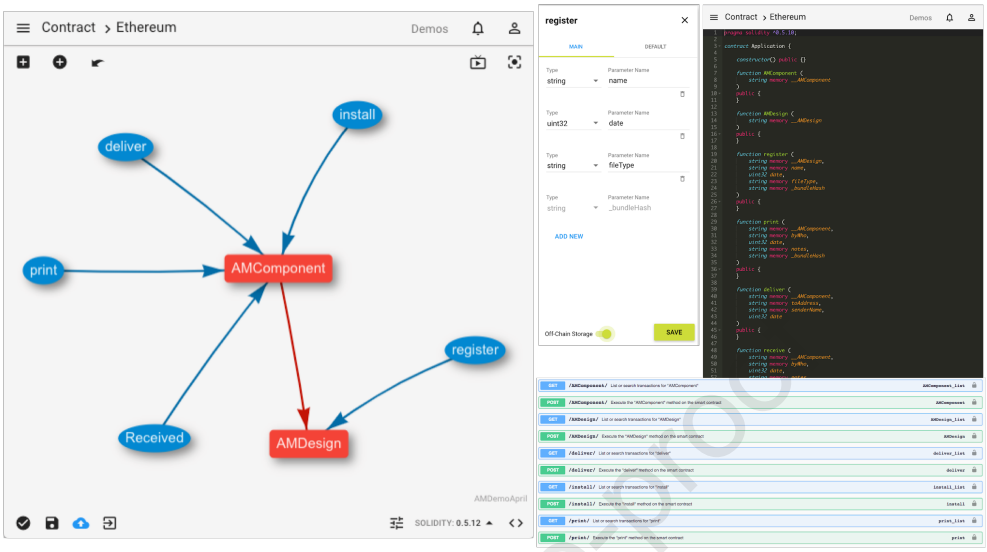

Figure 9: Illustrates the smart contract designer, the Solidity code that is generated, and the resulting API.

- To use the zero-code user interface to specify these graphically and auto-generate the corresponding Solidity code.
- To create a SIMBA app, which will deploy the smart contract and auto-generate a REST API to those business rules.

Where an asset is a physical entity, a document, a dataset, a binary, or media elements on the blockchain and a transaction points to a previously recorded asset and provides new information, typically updating its state.

Collectively, this conceptualizes the business data relationships surrounding the Tailhook, or a part of a Tailhook, which are tracked from removal to repair completion. The model can be specified using a simple Web-based Graph-style user interface, shown on the left part of Figure 9, which enables a user to specify assets, relationships between assets, transactions that record the state of an asset, and parameters that define their payloads, as shown on right. Upon completion, this model is converted into smart contract code for the selected language and version, e.g., Solidity for Ethereum (see right part of Figure 9). Once a smart contract is saved, the developer can then deploy it within a SIMBA App, which performs two functions. First, it takes the Smart Contract and generates a schema and REST API that represents the various methods and structures in the smart contract (in lower part of Figure 9). Methods and method parameters are stored in a format compatible with

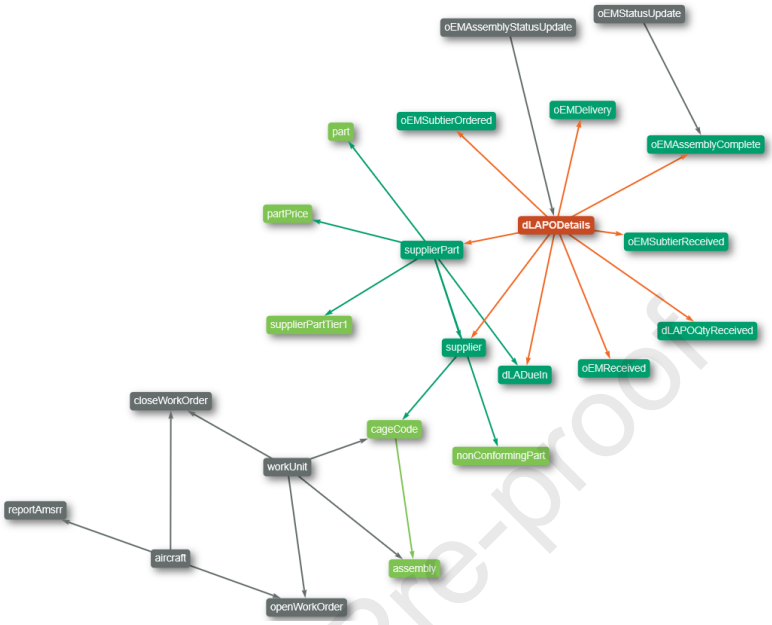

Figure 10: The OEM, NAVAIR, and GIDEP smart contracts in the SIMBA explorer

the Open API specification used by Swagger. Second, it deploys the Smart Contract onto the target desired blockchain network

The API is virtual and created when a developer hits the endpoint, which results in the API and Swagger documentation for the contract being auto-generated on the fly. Each smart contract method is a REST endpoint with two methods: a POST receiving a payload containing parameters for the method of a blockchain transaction; and a GET providing a mechanism of retrieving and searching multiple transactions, using query strings specific to the types of data being stored.

Ledger data belonging to two more smart contracts from different organizations, or even blockchains, can be linked together through any data points they have in common. A graph representing three smart contracts that have been linked together is shown in Figure 10. The OEM smart contract shows three assets that represent an order, a part from an order, and a supplier for that part. The transactions represent state changes along the pipeline. For example, at the OEM, there are four transactions that record when an OEM received the order, when a sub-tier order is placed for that part, when the part is received from the sub-tier, and when it is submitted to NAVAIR.

At NAVAIR, we record when parts are ordered and when parts arrive, along with the quantity received.

For the NAVAIR part, the graph shows the relationships between the MRO tracking stages of an assembly. This shows how a Tailhook assembly relates to a supplier, if ordered. And then shows the relationships between the work that is being performed on the Tailhook through the standard process of open and closed work orders, work units and how those relate back to the aircraft it belongs to, and further details of that aircraft. The GIDEP graph shows parts relating to pricing data and to a list of nonconforming parts or parts that are suspected of being counterfeit.

All three smart contracts have been linked together by two common data points. The resulting graph shows the relationship that can be explored between the different organizations and ledgers. For example, OEM and NAVAIR both link to a supplier asset. Thus, we can trace maintenance and repair work at NAVAIR to OEM suppliers. OEM, NAVAIR, and GIDEP all link to a cage code, the government identifier for a supplier. This extends traceability from work orders at NAVAIR to OEM suppliers to nonconforming parts and pricing data.

7.4. Deployment

We are deploying the SIMBA Chain platform using a Red Hat Enterprise Linux 7.9 Virtual Machine that will host a Hyperledger Fabric peer node and Quorum. An Ansible [43] Playbook enables rapid system configuration preparation and application deployment to vastly simplify the highly intricate process of configuring a Fabric node. Ansible is a configuration management, application deployment, and orchestration automation tool developed and supported by both the Ansible open source community and Red Hat Inc. There is a SIMBA NAVAIR software repository that contains the hyperledger specific configuration parameters for a Hyperledger fabric network, which can be used to deploy the software and perform a number of tasks in an idempotent fashion including: enable the Red Hat Enterprise Linux 7 software repositories; ensure a docker use group exists; ensure the docker user admin is added to the docker group; install necessary software packages on the host server; add a private SSH Key to access SIMBA private GitHub repositories; git clone the repository; ensure the docker daemon is started; ensure all 3rd party docker images are downloaded and available; download and unzip the Hyperledger Binaries from tarball to temporary directory; copy Hyperledger binaries to /usr/local/bin; modify permissions to Hyperledger

binaries and set as executable; and finally removed the tarball and clean up the temporary directory.

8. Retrospective and Insights

During the requirements analysis for this use case, our sponsor requested that we identify current gaps to identify extensions needed to support large scale production applications at scale for use cases similar to the one in this paper. This section therefore provides some gaps we extracted during this work to provide some insight into the needs for future production level systems. We also provide some more details on the blockchain systems we reviewed and some closing thoughts on enterprise oriented smart contract engineering and integration.

8.1. Blockchain Gap Analysis

Most of the gaps we identified in the blockchain layer involve the difficulties surrounding consortium-based blockchains, and issues with protecting, sharing and identifying data. The following list provides current gaps with the bullets outlining what we anticipate is needed in the future:

1. There are only a handful of confidentiality and authorization models that currently exist, which target specific scenarios:
   (a) More flexible models for on chain protection and authorization of transactions will be needed.
   (b) CAC Card-based authentication is needed for DoD blockchain enterprise integrations.
2. Identifiability and verifiability of asset and owners is not solved:
   (a) Establishing verifiability of data and owners will be needed, i.e., how does one know a chip is authentic and was manufactured by a specific entity? Emerging standards, such as Decentralized Identifiers [44] and Verifiable Credentials [45] may come into play here as they provide the capability of attesting claims about data or an entity using verifiable mechanisms. Such mechanisms can provide trust at the edge.

The main gaps for topology have to do with scaling consensus algorithms to increase throughput:

1. Transaction throughput is still relatively low compared to more traditional storage. Most studies indicate that a few thousand blockchain transactions per second is achievable right now:
    (a) Transaction throughput of consensus algorithms need to scale to the hundreds of thousands of transactions per second.
2. Enterprise blockchains currently use PoA style consensus models, but these protocols only partially address consortium-style blockchains:
    (a) Future algorithms could be more aligned to the participants' roles in the network. For example, emerging algorithms that come to a decentralized arbitration based on community voting might offer better security, reliability, and speedup over existing approaches.

For interfaces, we anticipate support for data specification, and even for blockchain updates will be needed:

1. Application specifications using smart contracts will change, and since smart contracts are non repudiable, versioning issues will need to be managed. Currently, there is no standard way to manage this:
    (a) Standardized versions patterns will be needed for smart contracts and blockchains.
2. Transactions are indexed using smart contract identifiers, so versioning impacts search functions also:
    (a) Standardized protocols that provide application sustainability for search and data retrieval to support multiple smart contract versions are needed for interoperability.

8.2. Platform and Application Gap Analysis

For platforms, we identify the following gaps:

1. Platform support for enterprise abstractions and efficient, resilient, and sustainable integration currently are in early stages, including:
    (a) Support for enterprise-oriented smart contract integration
    (b) Scalable and robust transaction execution
    (c) Enterprise integration support for existing authentication and authorization infrastructures
    (d) Support for event-driven models for transaction notifications

(e) Off-chain data integration support

2. Currently, there are no standardized ways to integrate enterprise data stores, e.g., SQL Server, ERP, MES, etc., into blockchain

   (a) Common patterns and standards for ingesting data, generating hashes, and binding to blockchain systems

For the application layer, we identify the following gaps and how these could materialize in the future:

1. Smart Contracts define the interface and logic about how transactions can take place, including on-chain access rights, multiple signatory requirements, and state logic.

   (a) The ordering of transactions defines the rules behind the business processes for each transaction, specifying the conditions that need to take place at each stage of the process. Moving forward, what is needed is a general methodology of defining such interactions that can lead to state machines to lay out the set of interactions for any flow of events that can be recorded using the blockchain.

2. There is a general lack of community understanding of blockchain technology and what the key problems it solves are

   (a) DoD should develop organic government expertise in blockchain technology and partner with industry to create and develop such expertise in DoD.

8.3. Blockchain Systems Comparison

As part of this work, we constructed a table to compare features from a number of blockchain systems in Figure 11, which were provided here for completeness.

This list is not exhaustive, but it contains representative example blockchain systems from the key application areas. The columns focus on five main areas: the industry focus area that the blockchain addresses; the ledger type, i.e., whether it is private (permissioned) or public; the consensus algorithm the blockchain uses; the smart contract language that the blockchain supports; and finally, information about the governance of the blockchain. For the blockchain list, we aimed at providing examples from two core areas: those designed for private enterprise blockchain applications and those designed for public use. The latter category is further subdivided into those focusing on cryptocurrencies and those that are more generic.

	Industry focus	Ledger Type	Consensus Algorithm	Smart Contract Language	Governance
Bitcoin	Crypto	Public	Proof of Work	None	Bitcoin Devs
Public Ethereum	Cross Industry	Public	Proof of Work	Solidity (EVM)	Ethereum Devs
Binance	Exchange	Public	Proof of Staked Authority	Solidity (EVM)	Binance Management and Devs
RSK	Cross Industry	Public	Proof of Work	Solidity (EVM)	RSK Community
Libra	Financial Services	Hybrid	Byzantine fault tolerant (BFT)	Move	Libra Association
Quorum	Cross Industry	Private	Majority Voting	Solidity (EVM)	Ethereum Devs & JP Morgan
Hyperledger Fabric	Cross Industry	Private	Pluggable Framework	Go, Java, JavaScript and using the Burrow EVM, Solidity	Linux Foundation
Hyperledger Sawtooth	Cross Industry	Private	Pluggable Framework	Go, Java, JS, Python Rust using Sabre (WASM), or Solidity (Seth)	Linux Foundation
Hyperledger Burrow	Cross Industry	Private	Proof Of Stake	Solidity (EVM) and C/C++ (WASM)	Linux Foundation
R3 Corda	Financial Services	Private	Pluggable Framework	Kotlin	R3 Consortium
Ripple	Financial Services	Private	Probabilistic Voting	Javascript (Codius)	Ripple Labs
Stellar	Financial Services	Public	Stellar Consensus Protocol	No language, but can define flows using Stellar Smart Contracts	Stellar Dev Foundation
EOS	Cross Industry	Private	Delegated Proof-of- Stake	C/C++ (WASM)	EOSIO Core Arbitration Forum
OpenChain	Digital Asset Management	Private	Partioned Consensus	Java, Clojure, Groovy, Kotlin (JVM)	CoinPrism

Figure 11: Comparison of different blockchain networks

8.4. Enterprise Oriented Smart Contract Integration

Smart contracts are nothing else but software and therefore need to be subjected to the same best practices that all enterprise software should be subjected to. This includes integrating smart contract development with existing development processes already in the enterprise, including version control, peer review, unit and integration testing, and continuous development and integration pipelines. Additionally, smart contracts, in particular should be subjected to security audits to ensure they are safe for blockchain execution. Finally, code that is deployed to the blockchain should be verifiable and itself expose a provenance trail that allows the consortium members executing transactions with the contract to have confidence in the code.

Smart contracts will evolve over time, like all software and APIs. EBPs need to allow contract lifecycle management including contract versioning, contract deprecation, API migration, management and insight into versions of a deployed contract, as outlined as a gap in interfaces (1a) in Section 8.1. Similarly, blockchains are evolving and therefore the provenance of contracts across different blockchains, is crucial to the management of contract lifecycle.

Smart Contracts and blockchains differ in their capabilities, data types and deployment scenarios. The EBP needs to provide an abstract layer that not only allows migration to different underlying networks but also ex-

poses APIs that are enterprise-centric, i.e. allow an enterprise to *express its business processes* through blockchain transactions. Rather than an enterprise being constrained by the particularities of an underlying blockchain infrastructure, the EBP should expose the blockchain using tools, interfaces and workflows that are familiar, well understood, and easily integrated into existing systems. In the same way that creating transactions should seamlessly integrate with existing workflows, so should search and query. Blockchains are only useful across multiple consortium partners if visibility into the blockchain provides easy, fast and rich search functionality allowing slicing and dicing the data along multiple dimensions.

9. Conclusions

In this paper, we've provided a comprehensive overview of a blockchain application that spans two different blockchain systems and connects the DoD with OEM suppliers in order to track the status of parts. We described the MRO use case in detail and extracted the technical requirements needed. We then performed a critical analysis of the state of the art of blockchains to discuss issues across the different layers in the system, and identify tools that can help meet the use cases requirements.

We then described an architecture for the solution and during the design sections, we provided the justification for using the SIMBA Chain platform for the development phase of the project. For development, we showed how the platform was configured to support the Quorum and Fabric blockchains to provide clear separability and privacy of data while enabling a GraphQL queryable interface to enable users at NAVAIR to query the data across both blockchains. We described the smart contracts and purposes and discussed the technology we used to deploy this as a package.

Even though this use case is simple and at the pilot phase, it shows many of the issues that need to be tackled in order to create blockchain integrations and deployments in real world use cases. We hope that the requirements of this use case along with the architecture, design, and implementation details, and gap analysis that may be needed to satisfy and scale such use cases in the future, provide insight to researchers as they start to innovate in the areas we have identified that provide an opportunity for improvement.

10. Acknowledgements

This work was made possible by funding from a Phase II SBIR grant (Contract No. N68335-20-C-0302) for the VIPART project (PO, Dr Manivannan Venkat) to support NAVAIR.

11. References

[1] G. Wood, Ethereum: A secure decentralised generalised transaction ledger - e94ebda (2018).
URL https://github.com/ethereum/yellowpaper

[2] Ethereum Transaction History.
URL https://etherscan.io/chart/tx

[3] Enterprise Ethereum Alliance.
URL https://entethalliance.org

[4] Consensys Quorum.
URL https://consensys.net/quorum/

[5] The Hyperledger open source community for enterprise-grade blockchain deployments.
URL www.hyperledger.org

[6] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, J. Yellick, Hyperledger fabric: A distributed operating system for permissioned blockchains, in: Proceedings of the Thirteenth EuroSys Conference, EuroSys '18, Association for Computing Machinery, New York, NY, USA, 2018. doi:10.1145/3190508.3190538.
URL https://doi.org/10.1145/3190508.3190538

[7] SIMBA Chain.
URL https://simbachain.com/

[8] N. B. Barnas, Blockchains in national defense: Trustworthy systems in a trustless world, Air University (U.S.). Air War College.
URL https://www.hsdl.org/?abstract&did=825348

[9] The 2018 National Defense Authorization Act .
URL https://docs.house.gov/billsthisweek/20171113/HRPT-115-HR2810.pdf/

[10] D. Woodfield, The emerging impacts of blockchain technology on dod asset cyber security, Air University (U.S.). Air War CollegeAccession Number: AD1107534.
URL https://apps.dtic.mil/sti/pdfs/AD1107534.pdf

[11] V. Adams, M. Alonso, W. Henry, D. Hyland-Wood, W. C. Jansen, V. Kodumudi, A. Nannra, J. Neidig, N. S. Matt Nelson, J. Stevens, W. H. Sutherland, I. Taylor, J. Tennenbaum, Potential uses of blockchain by the u.s. department of defense, On Line White Paper.
URL https://www.scribd.com/document/462322189/Paper-Potential-Uses-of-Blockchain-Technology-in-DoD

[12] L. M. De Rossi, N. Abbatemarco, G. Salviotti, Towards a comprehensive blockchain architecture continuum., in: Proceedings of the 52nd Hawaii International Conference on System Sciences, 2019.

[13] S. Haber, W. S. Stornetta, How to time-stamp a digital document, J. Cryptology 3 (1991) 99–111. doi:10.1007/BF00196791.

[14] The Bitcoin Foundation.
URL https://bitcoinfoundation.org/

[15] Ethereum.
URL https://ethereum.org/

[16] Stellar - an open network for money.
URL www.stellar.org

[17] RSK.
URL www.rsk.co

[18] Hedera Hashgraph.
URL www.hedera.com

[19] A exhaustive list of Cryptocurrencies.
URL https://coinmarketcap.com/all/views/all/

[20] Hyperledger Fabric.
URL https://www.hyperledger.org/use/fabric

[21] Hyperledger Burrow.
URL https://www.hyperledger.org/use/hyperledger-burrow/

[22] Ripple.
URL https://www.ripple.com/

[23] R3 Corda.
URL https://www.r3.com/

[24] Gartner's trough of disillusionment.
URL https://www.gartner.com/en/documents/3887767/understanding-gartner-s-hype-cycles

[25] Top 5 Blockchain Platforms.
URL https://www.hfsresearch.com/blockchain/top-5-blockchain-platforms_031618/

[26] 5 Best Blockchain Systems For Enterprises.
URL https://medium.com/swishlabs/the-5-best-blockchain-platforms-for-enterprises-and-what-makes-them-a-good-fi

[27] S. et al, gpi real-time nostro proof of concept (2018).
URL https://www.swift.com/sites/default/files/documents/swift_report_nostro_public_release_.pdf

[28] C. Cachin, A. De Caro, P. Moreno-Sanchez, B. Tackmann, M. Vukolic, The transaction graph for modeling blockchain semantics., IACR Cryptol. ePrint Arch. 2017 (2017) 1070.

[29] C. G. Akcora, Y. R. Gel, M. Kantarcioglu, Blockchain: A graph primer, arXiv preprint arXiv:1708.08749.

[30] The Graph Documentation.
URL https://thegraph.com/docs/

[31] Any Block Analytics.
URL https://www.anyblockanalytics.com/docs/main/

[32] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley Longman Publishing Co., Inc., USA, 2002.

[33] Truffle Suite.
URL https://www.trufflesuite.com/

[34] Remix.
URL https://remix.ethereum.org/

[35] Kaleido.
URL https://www.kaleido.io/

[36] Chain Link.
URL https://chain.link/

[37] MultiBaas.
URL https://www.curvegrid.com/

[38] Settlemint.
URL https://settlemint.com/

[39] Codefi Orchestrate.
URL https://codefi.consensys.net/orchestrate

[40] Hyperldger Besu.
URL https://besu.hyperledger.org/

[41] G. Foundation, The graphql specification, On Line White Paper.
URL http://spec.graphql.org/draft/

[42] Swagger and the OpenAPI Specification.
URL https://swagger.io/docs/specification/about/

[43] Ansible.
URL https://github.com/ansible/ansible

[44] D. Reed, M. Sporny, M. Sporny, D. Longley, C. Allen, R. Grant, M. Sabadello, J. Holt, Decentralized identifiers (dids) v1.0, core architecture, data model, and representations (November 2019).
URL https://www.w3.org/TR/did-core/

[45] M. Sporny, G. Noble, D. Longley, D. C. Burnett, B. Zundel, Verifiable credentials data model (November 2019).
URL https://www.w3.org/TR/vc-data-model/

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: