Adopting Hyperledger Fabric Blockchain for EPCglobal Network[1]

Pamela Hui Ting Chua
School of Information Systems
Singapore Management University
pamela.chua.2016@mais.smu.edu.sg

Yingjiu Li
School of Information Systems
Singapore Management University
yjli@smu.edu.sg

Wei He
Singapore Institute of Manufacturing Technology (SIMTech)
whe@simtech.a-star.edu.sg

ABSTRACT

The EPCglobal Network is a computer network used to share product data between trading partners. The EPC Information Services (EPCIS) is an event record repository that allows disparate applications to access and query for data both within and across enterprises. Ultimately, this data sharing is aimed at enabling participants in the EPCglobal Network to gain a shared view of the disposition of EPC-bearing objects within a relevant business context [1].

Despite the potential benefits, enterprises are reluctant to integrate into the EPCglobal Network due to the financial and manpower resources required to set up, operate and maintain the EPCIS and to adopt the standards. Issues like security, computation and storage overheads would also have to be managed. To solve the problem, this paper advocates a blockchain solution using Hyperledger Fabric to serve as a shared EPCIS across trading partners. The blockchain can be queried directly to facilitate information sharing subjected to access control rules. It also adheres to the EPCglobal Network standards. This approach provides several important benefits including eliminating the need for enterprises to maintain their own EPCIS while still being able to participate and benefit from the network. As the blockchain can be queried directly, the provision of EPC Discovery Services (EPCDS) may also be unnecessary. To explore this idea, we constructed a prototype using Hyperledger Composer and Hyperledger Fabric while adhering to the data elements, structures and formats as stated in the standards to ensure interoperability with existing architecture both upstream and downstream.

CCS Concepts

• Information systems →Enterprise applications; • Security and privacy →Distributed systems security

Keywords

Blockchain, Hyperledger, EPCglobal, EPCIS, RFID, Supply Chain, GS1

1. INTRODUCTION

The EPCglobal Network defines information systems, communication protocols, and data types that support capturing, storage and exchange of EPC data among supply chain network participants. Figure 1 depicts the standards in EPCglobal Network Architecture Framework. The architecture includes specification for low-level communication protocols such as the air interface between tag and reader as well as high-level aggregated business information such as the EPCIS and the EPC Discovery Service (EPCDS) [2]. EPCIS stores historical data and this data is not only raw data captured from data carriers, but also in contexts that imbue those observations with meaning relative to the physical or digital world and to specific steps in operational or analytical business processes [4]. Given an EPC number of an item in a supply chain, the EPCDS provides pointers to the resources that contain the read events created during the travel of the item through the supply chain, as shown in Figure 2.

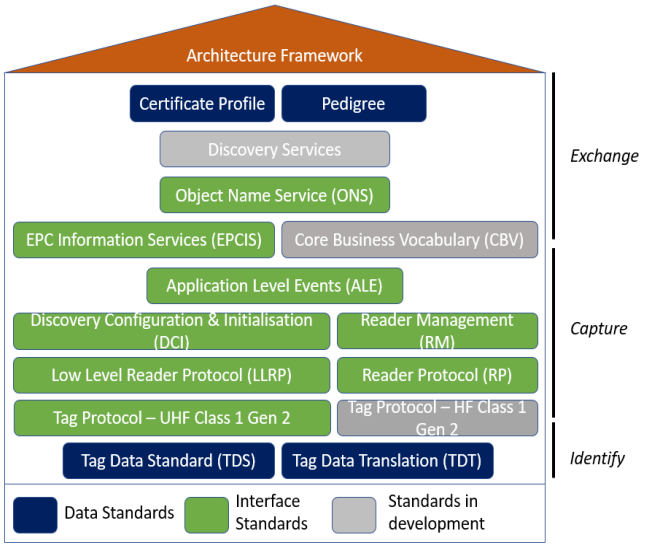

Figure 1: EPCglobal Architecture Framework [2]

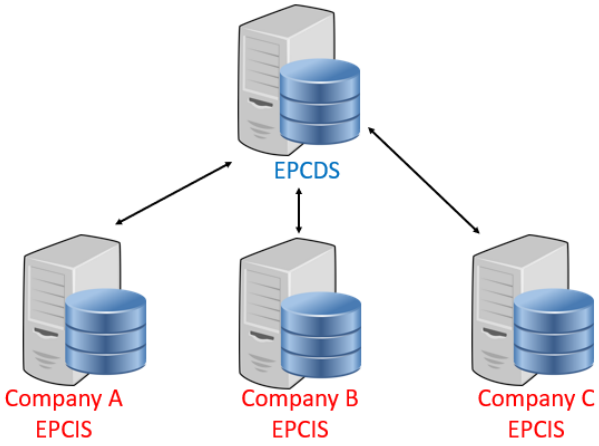

Figure 2: EPCIS and EPCDS

[1] 978-1-7281-1210-7/19/$31.00 ©2019 IEEE

978-1-7281-1210-7/19/$31.00 ©2019 IEEE

In the existing architecture, each trading partner will set up its own EPCIS. For mid or smaller sized companies, this is not easy due to developing cost and/or the integrating cost to the EPCglobal Network. Additional resources will also need to be allocated for the operation and maintenance of the system. Besides this, there are also security concerns such as the assumption of the EPCDS being a trusted entity. It also presents itself as a single point of failure in the network.

In this project, we aim to devise a blockchain solution that can overcome these limitations and still fulfil the functions of EPCIS and EPCDS.

2. BACKGROUND & PRELIMINARIES

2.1 GS1 Standards and EPCglobal Network

GS1 standards may be divided into three groups according to their role in supporting information needs related to real-world entities in supply chain business processes [4]:

1) Identification Standards.

GS1 identification standards which provide the means to **identify** real-world entities so that they may be the subject of electronic information that is stored and/or communicated by end users.

2) Data Capture Standards

GS1 data capture standards which provide the means to automatically **capture** data that is carried directly on physical objects, bridging physical things and electronic information.

3) Information Sharing Standards

GS1 standards for information sharing that provide the means to **share** information, both between trading partners and internally, providing the foundation for electronic business transactions, electronic visibility of the physical or digital world, and other information applications. Both the EPCIS and Core Business Vocabulary(CBV) standards are under this group and additionally, it includes discovery standards that help locate where relevant data resides across a supply chain and trust standards that help establish the conditions for sharing data with adequate security.

The primary type of data exchanged in the network are *read* events. They are business-level events, which represent a scan of an RFID tag or 2D barcode associated with business context. We aim to leverage on the features in Hyperledger Composer and Hyperledger Fabric to address the needs in information sharing (Group 3), adhering to specified standards, mainly the EPCIS and CBV. Referring to Figure 3, our solution should provide an alternative to the EPCIS repository and work with existing infrastructure like the EPCIS Capturing Interface and EPCIS Query Interface. The events are recorded via the capturing interface and stored in the Hyperledger Fabric blockchain, which can in turned be queried by users through the query interface.

2.2 GS1 EPCIS and CBV Standards

EPCIS is a GS1 standard that enables trading partners to share information about the physical movement and status of products as they travel throughout the supply chain. The EPCIS standard specifies the data elements in an EPCIS event and the CBV provides identifiers that may be used as values for those data elements. The CBV standard provides definitions of data values that may be used to populate the data structures defined in the EPCIS standard.

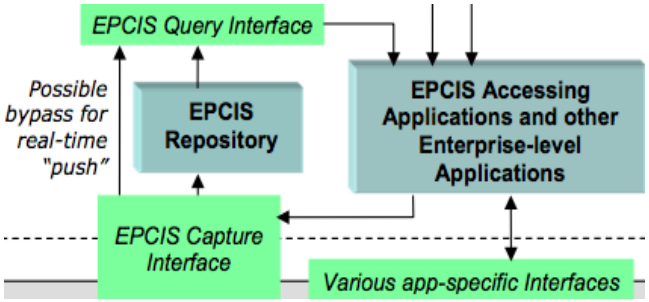

Figure 3: The interface between the "Capture" and "Share" standards

1) EPCIS Standard 1.2 [4]

Release 1.2, dated Sep 2016, is the latest version at the time of this paper. The goal of EPCIS is to enable disparate applications to create and share visibility data, both within and across enterprises. This sharing is aimed at enabling users to gain a shared view of physical or digital objects within a relevant business context. The EPCIS specification specifies only a standard data sharing interface between applications that capture visibility event data and those that need access to it. It does not specify how the service operations or databases themselves should be implemented.

Our focus in this standard is on the core event types. They are *ObjectEvent, AggregationEvent, TransactionEvent* and *TransformationEvent.* Each of the core event type (excluding the generic *EPCISEvent*) has fields that represent the four dimensions of any EPCIS event, namely "what, where, when and why". An EPCIS event is at a semantically higher level as it incorporates an understanding of the business context in which the identifier data were obtained.

2) CBV Standard 1.2.2 [3]

Release 1.2.2, dated Oct 2017, is the latest version at the time of this paper. The goal of this standard is to specify various vocabulary elements and their values for use in conjunction with the EPCIS standard. Using the vocabulary identifiers and definitions in this standard will ensure that all parties who exchange EPCIS data will have a common understanding of the semantic meaning of that data.

There are two kinds of vocabularies: (1) *Standard Vocabulary*, a set of Vocabulary Elements whose definition and meaning must be agreed to in advance by trading partners who will exchange events using the vocabulary. (2) *User Vocabulary*, a set of Vocabulary Elements whose definition and meaning are under the control of a single organisation. The master data associated with User Vocabulary elements are typically defined by end user organisations and are usually distributed to trading partners through the EPCIS Query Interface or other data exchange/data synchronisation mechanisms. Figure 4 and 5 provide a summary of the two vocabularies defined in the standard for a **CBV-Compliant Document**.

Vocabulary	Vocabulary Elements: Prefix	Optional
BusinessStepID	urn:epcglobal:cbv:bizstep:string	No
DispositionID	urn:epcglobal:cbv:disp:string	Yes
BusinessTransactionTypeID	urn:epcglobal:cbv:btt:string	Yes
SourceDestTypeID	urn:epcglobal:cbv:sdt:string	Yes
ErrorReasonID	urn:epcglobal:cbv:er:string	Yes

Figure 4: Standard vocabularies

Vocabulary	Vocabulary Elements: Prefix	Optional
EPCClass	• EPC URI • Private or Industry-wide URN e.g. *urn:URNNamespace:…*	Yes
ReadPointID		Yes
BusinessLocationID		Yes
BusinessTransaction		Yes
SourceDestID		Yes

Figure 5: User vocabularies

2.3 Hyperledger Fabric and Hyperledger Composer

Hyperledger Fabric is a permissioned blockchain framework that has a highly modular and configurable architecture, which allows components such as consensus and membership services to be plug-and-play. The distinctive characteristic about Fabric is that it allows entities to conduct confidential transactions through a private channel and data will only be shared among the selected participants. It is different from the public ledgers like Bitcoin blockchain and Ethereum as shown in Figure 6:

	Bitcoin	Ethereum	Hyperledger Fabric
Cryptocurrency-based	Yes	Yes	No
Permissioned	No	No	Yes
Pseudo-anonymous	Yes	No	No
Auditable	Yes	Yes	Yes
Immutable ledger	Yes	Yes	Yes
Modularity	No	No	Yes
Smart contracts	No	Yes	Yes
Consensus protocol	Proof of Work	Proof of Work	Various

Figure 6: Differences between Hyperledger Fabric and other permission-less blockchain technologies

Since the Fabric platform is permissioned, it means that unlike a public permission-less network, the participants are known to one other, rather than anonymous and untrusted. In Fabric, while the participants may not fully trust one other, a network can be operated under a governance model based on a legal agreement, contract, or framework for handling disputes. Other than that, there is also support for pluggable consensus protocols that enable the platform to be more effectively customized to fit particular use cases and trust models. Fabric enables confidentiality through its channel architecture. Only nodes that participate in a channel have access to the smart contracts and data transacted.

Hyperledger Composer, which is illustrated in Figure 7, is a set of tools that allows users to quickly built, test and manage blockchain applications. It can be used to model business networks and supports the existing Hyperledger Fabric blockchain and runtime. It offers business-centric abstractions and makes it simple and fast to create smart contracts and blockchain applications to solve business problems.

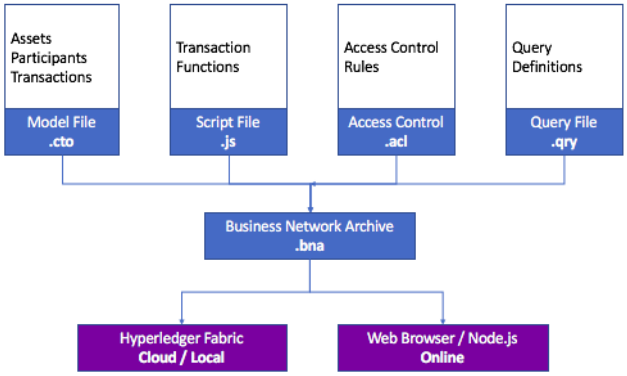

Figure 7: Hyperledger Composer Framework [13]

As part of the business network model, we define the assets followed by the transactions that can interact with the assets. Business networks also include participants who can interact with the assets, each of which can be associated with a unique identity, across multiple business networks. These participants can have their access to transactions restricted based on their roles. Queries can be used to filter results returned using criteria and can be invoked in transactions to perform operations. The Hyperledger Fabric instance acts as the underlying shared database for the business network. A web browser or a front-end application can be developed to interact with the network.

3. THE EPCIS BUSINESS NETWORK MODEL IN HYPERLEDGER COMPOSER

The EPCIS network is a business network that consists of a Manufacturer, a Shipper, an Importer and a Distributer class. The specifics are spelled out in the README.md file. The network is modelled using the CTO modelling language (Hyperledger Composer Model Language) into five model files, with the business logic enforced by chaincode (smart contract) written in Javascript. The EPCIS and CBV standards were strictly adhered to and integrated into the network to ensure interoperability with existing infrastructure.

3.1 Model Files

The following five model files together represent the outline and structure of the network and its entities. They contain the definitions of each class of asset, transaction, participant, and event.

3.1.1 epcis.cto

The epcis.cto file defines the EPCIS events as assets in the business network. An abstract type *EPCISEvent* is declared and is used as a basis for the other event classes to extend from. There are four event classes namely *ObjectEvent, AggregationEvent, TransactionEvent* and *TransformationEvent* as defined in Section 7.4 of [4].

According to Section 8.8 of [3], an event identifier may populate the *eventID* field of an EPCIS event. It was stated that eventID is not a compulsory field but when it is included, the identifier must be globally unique (different from the event

identifier in any other EPCIS event created by any party). However, in our Hyperledger Composer implementation, *eventID* is made a compulsory field and serves as the unique identifier for EPCIS events. Following the CBV-Compliant requirement, we use UUID Version 1 or Version 4 URI as specified in [RFC4122] to populate the *eventID* field. Regular expression is used to enforce this as shown in Figure 8.

```
abstract asset EPCISEvent identified by
eventID {

    ...

    o String eventID regex = /^[0-9a-fA-F]{8}-
[0-9a-fA-F]{4}-([1]|[4]){1}[0-9a-fA-F]{3}-
[89abAB]{1}[0-9a-fA-F]{3}-[0-9a-fA-F]{12}$/

    ...

}
```

Figure 8: Use of regex to enforce UUID Version 1 or Version 4 URI for *eventID* field

Several attributes of EPCIS events e.g. *ErrorDeclaration* and *BusinessTransaction* are structures, also known as concepts, which are further defined in reusabletypes.cto. *ErrorDeclaration* is an optional element. An event containing this element shall be identical to a prior event and indicates that all the assertions implied by the prior event are erroneous, as of the specified *declarationTime* in the *ErrorDeclaration* structure. This is necessary as EPCIS events can neither be deleted nor updated. Every *EPCISEvent* asset should also be tagged to a network participant.

Other than the attributes inherited from *EPCISEvent*, each of the core event type also contains its own unique fields. For example, unlike other event types, *TransactionEvent* requires at least one *BusinessTransaction* object. *ObjectEvent* also contains two unique fields, namely *epcList* and *quantityList*. The *epcList* and *quantityList* cannot be both empty at the same time. Such logic shall be implemented by the transaction processor functions in the script files.

3.1.2 participants.cto

The participants.cto file defines the participants in the business network. An abstract type *Business* is declared and is used as a basis for the network participant classes to extend from. There are four participant classes in this business network, namely Manufacturer, Shipper, Importer and Distributer. They are identified by the *companyId* field and contain basic properties such as address, email and company name. EPCIS events are tagged to participants that own the corresponding capturing interface or execute the create EPCIS event transaction. These events are stored as an array and can be referenced through the respective participant.

3.1.3 masterdata.cto

The masterdata.cto file models the vocabularies and master data attributes defined in the CBV for a CBV-Compliant Document. Prefixes of the values for each data element under Standard Vocabulary may be enforced using regex as shown in Figure 9.

```
asset BusinessStepID extends MasterData {

    o String desc regex =
/^(urn:epcglobal:cbv:bizstep:)/

}
```

Figure 9: Use of regex to enforce value prefix for bizstep field

Besides prefix, according to the CBV, the suffix of the element values must also adhere to those specified in the document. In the case of bizstep, some of the allowed suffix values are *accepting, arriving, assembling, collecting, commissioning, consigning* etc. For data elements with more than three possible suffix values, regex will not be used for pattern-matching to keep the file simple. *ErrorReasonID* has two possible suffix values and regex can be implemented as shown in Figure 10.

```
asset ErrorReasonID extends MasterData {

    o String desc regex =
/^(urn:epcglobal:cbv:er:did_not_occur|incorrec
t_data)$/

}
```

Figure 10: Use of regex to enforce suffix values for error reason field

Unlike standard vocabularies, user vocabularies are created by an end user and their meanings may be described to trading partners via master data exchange, or via some other mechanism outside of the EPCIS Query Interface. Hence, user vocabularies are modelled slightly differently and subjected to different access control rules. Nevertheless, the syntax of identifiers for user vocabularies is still required to follow a predicted structure according to the CBV. However, due to the numerous variations such as EPC URI, HTTP URL and private or industry-wide URN, the values for user vocabularies data elements will not be enforced with regex in masterdata.cto.

Typically, master data attributes are specified within the master data section of the EPCIS header in an EPCIS Master Data Document or in the response to an EPCIS Master Data Query. In our Hyperledger Composer implementation, the master data attributes are tagged to the respective user vocabularies.

3.1.4 reusabletypes.cto

The resusabletypes.cto file contains structures or objects that are frequently referenced to in the EPCIS events. Examples include ILMD, QuanityElement, ErrorDeclaration etc.

3.1.5 transactions.cto

The transactions.cto file models the following transactions in the EPCIS business network:

1) *SetupEPCISEvents* – This transaction is executed to populate the Asset Registries with four EPCIS sample events of each core type which are tagged to network participants.
2) *SetupVocab* – This transaction is executed to populate the Asset Registries with vocabulary assets of ten different classes.
3) *SetupParticipants* – This transaction is executed to populate the Participant Registries with five *Business* participants namely a *Manufacturer*, two *Importer*, a *Shipper* and a *Distributer*.

4) *CreateEPCISEvent* – This is an abstract transaction that can be extended to create events of each core type. It cannot be executed on its own. It is used for both new and error correction events.
5) *CreateObjectEvent* – This transaction simulates the creation of an Object Event. It extends *CreateEPCISEvent* and serves as an example to the creation of other core event types as besides the attributes that are slightly different, the process is generally similar.
6) *CreateUserVocabRecord* – This transaction simulates the creation of a user vocabulary record.
7) *UpdateUserVocabRecord* – This transaction simulates the update of a user vocabulary record.

3.2 Script Files

The scripts in a Hyperledger Composer Business Network Definition contain transaction processor functions and other supporting functions that implement the transactions defined in the model files. Transaction processor functions are automatically invoked by the runtime when transactions are submitted using the BusinessNetworkConnection API. In our business network, there are three script files implementing the logic as follows:

3.2.1 setupdemo.js

The setupdemo.js file contains three transaction processor functions that set up sample data for the EPCIS business network. The transactions shall be submitted in the following order:

1) *SetupParticipants* – Invoke function *setupParticipants* to set up five participants.
2) *SetupVocab* – Invoke function *setupVocab* to set up sample vocabularies and masterdata.
3) *SetupEPCISEvents* – Invoke function *setupEPCISEvents* to set up four EPCIS events.

The *setupParticipants* function sets up five sample network participants, namely one *Manufacturer*, one *Distributer*, two *Importers* and one *Shipper* and commits them to the Participant Registry by calling the *createParticipants* function.

The *setupVocab* function sets up sample data for all the ten classes of user and standard vocabularies. Each user vocabulary record is defined by and tagged to a network participant, whereas standard vocabulary does not belong to any specific participant and uses fixed values as stated in the standards. The *createData* function is called by the *setupVocab* function to commit the vocabularies into the respective Asset Registries.

Lastly, the *setupEPCISEvents* function sets up four EPCIS events, namely one Object Event, one Aggregation Event, one Transaction Event and one Transformation Event, each tagged to a network participant. It then calls the *createEvents* function to commit the events into the respective Asset Registries.

The setup of the sample data could not be completed together in one transaction as in our model design, both the vocabularies and EPCIS events are tagged to the participants. The current limitation in Hyperledger Fabric (as used by Hyperledger Composer) is that you cannot read your own writes i.e. cannot add something to a registry in a transaction and then read it in the same transaction. This is because the transaction has not been committed at this point and hence the entry cannot be read back from the world state in the same transaction. Hence in our case, the transaction *SetupParticipants* would need to be executed first to commit the network participants into the registry before the vocabularies and EPCIS events can be set up to link to the participants.

Figure 11 and 12 show how relationships can be established between Participant and Asset classes.

```
abstract asset EPCISEvent identified by
eventID {

    ...

    --> Business

}
```

Figure 11: An EPCIS event having a relationship to a network participant (epcis.cto)

```
asset ReadPointID extends MasterData {

    ...

    --> Business

}
```

Figure 12: A user vocabulary record having a relationship to a network participant (masterdata.cto)

```
...
const array = await registry.getAll();
for (let p = 0; p < array.length; p++) {
    if (array[p].desc == updateRec.olddesc) {
        // keep id the same
        rec = array[p];
        if (rec.business.getIdentifier() !=
        getCurrentParticipant().getIdentifier()
        ){
            throw "You do not have access to
            update!";
         }
        rec.desc = updateRec.newdesc;
        rec.datetimeUpdated =
        updateRec.timestamp;

        ...
```

Figure 13: Code snippet in *updateUserVocabRecord* function in masterdata-logic.js

3.2.2 masterdata-logic.js

This script file contains two transaction processor functions (1) *CreateUserVocabRecord*, to create a new user vocabulary record. The new record will be tagged to the network participant who executed the transaction. (2) *UpdateUserVocabRecord*, to update an existing user vocabulary record. Only the network participant who created the original record will be able to update it.

In the function *createUserVocabRecord*, we first check if the URI entered in the transaction already exists in the vocabulary registry. If it does, an error will be thrown. Else, an id will be generated randomly (a number between 0 and 299) and assigned to the URI. As id must be unique, the number generation will continue until an unused value is produced. The new record will then be tagged to the participant and committed to the registry.

In the function *updateUserVocabRecord*, we retrieve the registry record with *desc* field equals to the *olddesc* entered and replaced it with *newdesc*, provided that the participant executing the transaction is the same as that of the original record. This is to ensure that records can only be modified by the creator. An error will also be thrown if the URI entered in the *olddesc* field cannot

be located in the registry. The *datetimeUpdated* field will be populated with the transaction time. Figure 13 shows a snippet of the *updateUserVocabRecord* function that denies the vocabulary record to be modified by someone who's not the owner/creator of the record.

3.2.3 epcisevents-logic.js

This script file contains the transaction processor function for the *CreateObjectEvent* transaction. *CreateObjectEvent* serves as an example to the creation of EPCIS events. Creation of other EPCIS events e.g. Aggregation Event and Transformation Event would likely follow the same process other than slight differences in the properties or attributes.

The script first checks if the optional *errorDeclaration* field is true. This is to ascertain if the event received is a new event or an error declaration event to correct a previous erroneous one. It signifies all assertions implied by the prior event with the same *eventID* to be erroneous as of the specified *declarationTime.* As stated in the standards, EPCIS events can neither be modified nor deleted and hence, an error declaration event is used to rectify prior mistakes. In our implementation, we make *eventID* compulsory and a unique identifier for EPCIS events. If *errorDeclaration* is true, we first check if the *eventID* already exists in the registry and if it does, all other fields besides *recordTime* and *errorDeclaration,* should be identical to those in the prior event. An error declaration event is only valid if it fulfils the above-mentioned criteria or an error will be thrown. To commit a valid error declaration event into the registry, we add a suffix of "-E" to the *eventID* to represent the relationship to the prior erroneous event. This is a deviation from the standards wherein *eventID* must be the same for both. This is because Hyperledger Composer does not allow duplicates in identifier. The *eventTime* field will also be identical to that of the prior event, whereas *recordTime* and *declarationTime* will be filled with the transaction time.

If the *errorDeclaration* field is false, it means that the event generated is a new event. The script will check that the *eventID* entered does not yet exists in the registry and then create the new record. The *recordTime* will be the same as the *eventTime* as it is assumed that once the event is captured, it is directly recorded into the EPCIS repository. We also ensure that there cannot be both an empty *epcList* and *quantityList* for the Object Event and it shall not contain the *ilmd* field if the *action* is "OBSERVE" or "DELETE". The new record will be committed to the registry if all checks pass.

3.3 Access Control

Hyperledger Composer implements the concept of role-based security through permissions built into its architecture to handle both authentication and authorization. A participant's access to resources is controlled based on its issued identity. Access control rules to protect resources in the network are stated in the *permissions.acl* file.

3.3.1 permissions.acl

There are ten access control rules written for the EPCIS business network. The following is a summary:

1. Business network admins are allowed ALL access to both user and system resources, namely CREATE, READ, UPDATE and DELETE. However, they are unable to invoke the *CreateUserVocabRecord*, *UpdateUserVocabRecord* and *CreateObjectEvent* transactions due to chaincode logic. This blanket rule might need to be reviewed in a production environment for security.
2. Business network participants are allowed to READ all vocabularies.
3. Business network participants are allowed to CREATE and UPDATE their own user vocabularies through the *CreateUserVocabRecord* and *UpdateUserVocabRecord* transactions respectively. They are also able to DELETE their own user vocabularies. However, they cannot make any changes to standard vocabularies.
4. Business network participants are allowed to READ only their own EPCIS events.
5. Business network participants are allowed to CREATE Object Events through the *CreateObjectEvent* transaction. No other action is permitted as EPCIS events can neither be updated nor deleted.

4. DEPLOYMENT TO HYPERLEDGER FABRIC

4.1 Development Environment

An AWS EC2 Instance was set up to run the development environment for Hyperledger Composer and Hyperledger Fabric. The Instance Type is t2.medium and an elastic IP of 35.155.128.185 was associated to the instance. Programs as shown in Figure 14 were installed.

Program/Software	Version
Ubuntu Linux	16.04 LTS
Docker	v18.03.1-ce
Docker Compose	v1.13.0
Composer Command Line Interface (composer-cli)	v0.19.7
Node.js	v8.11.2
npm	6.1.0
Python	2.7.12

Figure 14: Development environment

After the EPCIS Business Network Definition is completed, a .bna file that defines the compiled logic of the blockchain can be created. The .bna file is then deployed onto the Hyperledger Fabric blockchain. To interact with the blockchain, we generate a Loopback-based REST interface. An Angular application can then interface with the running REST server to connect to the Fabric instance. Figure 15 illustrates the process:

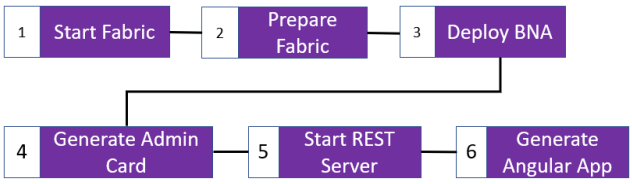

Figure 15: Deployment process

4.2 ID Cards

A Business Network Card provides all the necessary information to allow a participant to connect to a business network. Once issued, the user has an identity to be authenticated and authorized to use the network. Instead of using passwords, we import a card into a wallet and the card can be referenced to authenticate a user. This card can be shared with others, allowing them to connect to the business network using the issued identity. A Business Network Admin can create Business Network Cards for

participants in the context of the business network. The Business Network Admin in our network is *admin@epcis-blockchain*.

Deploying a business network to a Hyperledger Fabric instance for the first time requires that the Hyperledger Composer chaincode be installed on the relevant Hyperledger Fabric peers, and that the business network then be instantiated on the channel. This process requires special Hyperledger Fabric privileges possessed by a peer or channel administrator. This Peer Admin is also able to create, issue and revoke ID cards for Business Network Admins. Figure 16 shows the ID cards used in this business network. The Peer Admin for our Hyperledger Fabric instance is *PeerAdmin@hlfv1*.

```
ubuntu@ip-172-31-26-14:~$ composer card list
The following Business Network Cards are
available:

Connection Profile: hlfv1
```

Card Name	UserId	Business Network
Manufacturer_1@ epcis-blockchain	Manufacturer_1	epcis-blockchain
admin@epcis-blockchain	admin	epcis-blockchain
PeerAdmin@hlfv1	PeerAdmin	

```
Issue composer card list --card <Card Name> to
get details a specific card

Command succeeded
```

Figure 16: ID cards for EPCIS business network

4.3 Installation of Business Network

Deploying a business network to the Hyperledger Fabric instance requires the network to be installed on the peer. After which, we will create a Business Network Admin card using our Peer Admin card and import the former to be used in the business network. Once the network is started, it can be pinged to check for response as illustrated in Figure 17. To interact with the network, we can access Composer Playground at http://35.155.128.185:8080/login.

```
ubuntu@ip-172-31-26-14:~/epcis-blockchain$
composer network ping --card admin@epcis-
blockchain

The connection to the network was successfully
tested: epcis-blockchain
        Business network version: 0.0.2-
deploy.200
        Composer runtime version: 0.19.7
        participant:
org.hyperledger.composer.system.NetworkAdmin#a
dmin
        identity:
org.hyperledger.composer.system.Identity#37411
f5991e85db7837ec8634a0a4b8d3b51998645e10a4770d
fdc07b4279e25

Command succeeded
```

Figure 17: Pinging business network

4.4 Composer REST Interface

The REST API is generated based on the business network and provides a useful layer of language-neutral abstraction. The REST server includes functionality that generates a set of RESTful APIs for all the assets, participants and transactions within a deployed business network. EPCglobal capturing applications will be able to interact with the APIs to record EPCIS events into the blockchain, while assessing applications can query the APIs to obtain information regarding specific events or items. Our rest server was generated and available on http://35.155.128.185:3000/explorer.

Further configuration can also be applied to the REST server such as enabling event publication or TLS security to make it more robust and suitable for production environment. For example, the REST server can be configured to authenticate clients. When this option is enabled, clients must authenticate to the REST server before they are permitted to call the APIs.

4.5 Angular Application

Hyperledger Composer can also generate a skeleton Angular 4 application running against the REST API. All the changes done in the app will be reflected in the Hyperledger Fabric instance. The application is available on http://35.155.128.185:4200. The application has basic features as the development of the front-end application does not fall within our defined scope of work.

Note that the AWS instance and the respective services will be up and running till 5 April 2019.

5. EVALUATION OF SOLUTION

In this project, we have written a supply chain network i.e. EPCIS business network using the Hyperledger Composer Modelling Language, comprising of 1 Manufacturer, 2 Importers, 1 Shipper and 1 Distributer. We then deployed the business network archive to an instance of Hyperledger Fabric running on AWS and generated a Loopback-based REST interface that we can use to interact with the sample network blockchain application. There is one Peer Admin and one Business Network Admin in our setup.

A real-life supply chain network will be more complex e.g. a manufacturer may work with multiple distributers and charge them different prices for the same goods. These differing prices should be kept secret from competitors and other clients. There must also be appropriate distribution of power when it comes to approving transactions, committing to the blockchain and changing of world state to prevent collusion between parties. These requirements can be fulfilled by many features of Hyperledger Fabric e.g. having multiple private channels whereby data from one cannot be accessed from another unless explicit permissions are given. There will also be more nodes playing the roles of Client, Peer or Orderer where (1) **Client**, submits an actual transaction-invocation to the endorsers, and broadcasts transaction-proposals to the ordering service. (2) **Peer**, commits transactions and maintains the state and a copy of the ledger. Peers can have a special endorser role. (3) **Orderer**, runs the communication service that implements a delivery guarantee, such as atomic or total order broadcast. Endorsement policies can be included to evaluate transactions whereby a transaction is only valid and committed if it obtains an endorsement from all/enough endorsing peers subjected to the policy.

During the development of the solution, real-world sanitized data provided by Singapore Institute of Manufacturing (SIMTech) has been cross-referenced to ensure that the solution adheres to existing standards and is suited for real-world implementation.

However, like many blockchain solutions, scalability is of concern due to the potential magnitude of EPCIS event records. An improvement to the paper would be to evaluate the performance of the solution with different sizes of dataset.

6. RELATED AND FUTURE WORK

SecDS: A Secure EPC Discovery Services System in EPCglobal Network [7] proposed a secure and efficient search engine (SecDS) based on EPC Discovery Services (EPCDS). Besides enabling supply chain partners to query for possession of an item, the most important property of SecDS is being secure while efficiently processing user's search. An extended attribute-based access control (ABAC) model is proposed for SecDS that enriches the expressiveness of access control policies, while supporting visibility policies. Traditionally, access control policy for EPCIS is simpler than that of EPCDS as policies for an EPCIS are defined by the administrator of the EPCIS only, whereas the policies for EPCDS are defined by different security administrators of different companies. In our paper, as we are proposing to have a shared EPCIS in the form of a Hyperledger Fabric blockchain and doing away with EPCDS, we can explore incorporating the SecDS ABAC model into the default Hyperledger Composer access control module to improve the expressiveness and control of the access policies.

Fine-grained Access Control for EPC Information Services [9] proposed a fine-grained access control mechanism to protect the information in EPCIS, where query modification is used. The access control mechanism is depicted as a logical component between the storage component and the Query Interface. In our current Hyperledger Composer implementation, the default access control mechanism is simply a file listing access control rules. In a production environment, a more effective and fine-grained access control might be required to query the Hyperledger Fabric blockchain, which is essentially a shared EPCIS repository. This improves the usability and reliability of the solution. The policy language AAL (Auto-ID Authorization Language) proposed in the paper may be adopted in our current solution for a more robust EPCIS blockchain implementation.

Lastly, a more comprehensive and larger-scale blockchain prototype that uses real-life data and based on an actual supply chain network may be attempted. There are also other features/modules of Hyperledger Composer and Hyperledger Fabric that can be plugged in and customized to build a more robust solution. As the Hyperledger Business Blockchain Technologies are still emerging and undergoing rapid development, there will be new and exciting features in the foreseeable future that will make Hyperledger more promising for business applications.

7. CONCLUSION

Using blockchain for supply chain is not a new concept and in fact, has been one of the most promising use cases thus far. A good example is Everledger [14] that leverages on Hyperledger Fabric to track and protect items of value such as Diamonds. It facilitates visibility and provenance along the supply chain and ensures the authenticity of the diamonds is secured and stored among all industry participants. It was able to create a digital thumbprint for individual diamonds.

However, many of the implementations we have seen involve tracking of items through the supply chain while not adhering to any particular data exchange format or standards. This reduces inter-operability between systems and might require companies to reinvent the wheel i.e. overhaul of existing infrastructure.

This paper approaches the implementation of blockchain to supply chain from a different angle. The scope of this paper is in finding ways to marry Hyperledger Composer/Fabric to EPCIS and CBV standards, such that we can leverage on existing Hyperledger features to comprehensively express the granularity in the standards. Using the framework comprising of model files, script files and access control files inherent to Hyperledger Composer, we designed, proposed and implemented the definition and logic of the business network and data structures after a series of experimentation. We comply to EPCglobal Network standards and seek to replace a traditional EPCIS repository with a blockchain. Instead of individual EPCIS repository maintained by every trading partner, we proposed using a shared distributed ledger to store all the EPC event records generated from the flow of goods in the business network. Trading partners can query for event records subjected to verification of identity and access control. Potential benefits of such an implementation include reducing costs for companies and improving integrity of data. Other than that, it also helps to do away with the need for an EPCDS which presents itself as a single point of failure. By adhering to existing standards, we are also able to interface with existing infrastructure and facilitate inter-operability.

8. REFERENCES

[1] A. Syed and M. Ilyas. RFID Handbook: Applications, Technology, Security, and Privacy. CRC Press, 2008.

[2] M. Lorenz, J. Müller, M. Schapranow, A. Zeier and H. Plattner (2011). Discovery Services in the EPC Network, Designing and Deploying RFID Applications, Dr. C. Turcu (Ed.), ISBN: 978953-307-265-4, InTech.

[3] GS1 Core Business Vocabulary Standard, Release 1.2.2, Ratified, Oct 2017.

[4] GS1 EPC Information Services (EPCIS) Standard, Release 1.2, Ratified, Sep 2016.

[5] EPCIS and EPCDS. http://www.mysmu.edu/faculty/yjli/RFIDsec/slides/Li-SecDS-CODASPY12.pdf

[6] Hyperledger: Blockchain for Business. https://medium.com/swlh/hyperledger-chapter-2-hyperledger-frameworks-modules-cabf50e12105

[7] J. Shi, D. Sim, Y. Li, R. Deng. SecDS: A Secure EPC Discovery Services System in EPCglobal Network. In ACM CODASPY 2012.

[8] Hyperledger Composer Transaction Processor Functions. https://hyperledger.github.io/composer/latest/reference/js_scripts

[9] E. Grummt and M. Muller. Fine-Grained Access Control for EPC Information Services. In IOT, pages 35–49, 2008.

[10] S. Beier, T. Grandison, K. Kailing, and R. Rantzau. Discovery Services-Enabling RFID Traceability in EPCglobal Networks. In COMAD, pages 214–217, 2006

[11] C. K¨urschner, C. Condea, O. Kasten, and F. Thiesse. Discovery service design in the EPCglobal Network: towards full supply chain visibility. In IOT, pages 19–34, 2008.

[12] Hyperledger Composer. Available from: https://hyperledger.github.io/composer/v0.19/installing/development-tools.html

[13] Hyperledger Composer. Available from: https://hyperledger.github.io/composer/latest/introduction/introduction.html

[14] Everledger. Available from: https://www.altoros.com/blog/a-close-look-at-everledger-how-blockchain-secures-luxury-goods/