Journal of Manufacturing Systems 56 (2020) 157–174

Contents lists available at ScienceDirect

Journal of Manufacturing Systems

journal homepage: www.elsevier.com/locate/jmansys

Decentralized cloud manufacturing-as-a-service (CMaaS) platform architecture with configurable digital assets

Mahmud Hasan, Binil Starly*

Edward P. Fitts Department of Industrial and Systems Engineering, North Carolina State University, Raleigh, NC, 27695, USA

ARTICLE INFO

Keywords:
Cybermanufacturing
Blockchain
Smart contracts
Distributed digital thread
Cloud manufacturing

ABSTRACT

Contemporary Cloud Manufacturing-as-a-Service (CMaaS) platforms now promise customers instant pricing and access to a large capacity of manufacturing nodes. However, many of the CMaaS platforms are centralized with data flowing through an intermediary agent connecting clients with service providers. This paper reports the design, implementation and validation of middleware software architectures which aim to directly connect client users with manufacturing service providers while improving transparency, data integrity, data provenance and retaining data ownership to its creators. In the first middleware, clients have the ability to directly customize and configure parts parametrically, leading to an instant generation of downstream manufacturing process plan codes. In the second middleware, clients can track the data provenance generated in a blockchain based decentralized architecture across a manufacturing system. The design of digital assets across a distributed manufacturing system infrastructure controlled by autonomous smart contracts through Ethereum based ERC-721 non-fungible tokens is proposed to enable communication and collaboration across decentralized CMaaS platform architectures. The performance of the smart contracts was evaluated on three different global Ethereum blockchain test networks with the centrality and dispersion statistics on their performance provided as a reference benchmark for future smart contract implementations.

1. Introduction

Increasing trends in digitalization [1], data-driven smart manufacturing [2], and the presence of highly scalable networked architectures have allowed manufacturing machines on shop floors to rapidly respond to consumer demands while reducing cost and time via agile manufacturing techniques [3]. With the introduction of Cyber-Physical Systems (CPS) [4–6] and the subsequent expansion of cloud computing and manufacturing capabilities [7–9], there has been a paradigm shift in the methods used by design and manufacturing service companies to conduct business. Increased digital proliferation and improved virtualization methods have also enabled manufacturing entities to easily connect to the cloud and enable low-latency production tracking [10]. Today, digital platforms have transformed into CMaaS platforms wherein resources that are dynamically scalable and virtualized are provided as a service over the internet [11]. CMaaS platforms have been able to assimilate soft and hard manufacturing resources into a manufacturing cloud [3,12] and act as continuously evolving, knowledge-based intelligent manufacturing centers.

CMaaS platforms through cloud manufacturing have been able to democratize manufacturing services by making esoteric manufacturing expertise more accessible to the end user by shifting much of the technological and intellectual overhead away from the user. The user can now receive an instant quote for manufacturing a part within seconds [13]. This has been enabled by integrated CPS in CMaaS platforms that exploit scalable architectures and communication methods allowing direct operation of manufacturing entities over the cloud [14] and promising best utilization of manufacturing resources [15]. This process has greatly reduced time required for small volume production of a part by reducing the time-consuming process of requesting for quotes (RFQ), receiving feedback, transferring part designs and so on. While these benefits of contemporary CMaaS platforms have been well accepted across academia and industry, there are still several challenges associated with such platforms to improve widespread adoption among end users. In a comprehensive study of information systems, Oliveira et al. [16] employed the Diffusion of Innovation theory [17] and the Technology-Organization-Environment perspectives [18] to assess the control factors that regulate the innovation diffusion and adoption of cloud manufacturing within organizations and end users. The findings of the research suggest that, the adoption of cloud manufacturing on a massive scale is inhibited by a plethora of factors namely complexity associated with disruptive technologies like cloud manufacturing,

* Corresponding author.
E-mail address: bstarly@ncsu.edu (B. Starly).

https://doi.org/10.1016/j.jmsy.2020.05.017
Received 31 December 2019; Received in revised form 25 May 2020; Accepted 29 May 2020
Available online 20 June 2020

technological unreadiness, lack of top management support and smaller organization sizes. Currently, the growth of instant quoting and instant connection to job shops services are also hindered by the fact that information asymmetry across the two sides prevents clients and service providers from easily knowing the identity of either side. This limits growth of such platforms into highly regulated markets (e.g., defense, aerospace, and medical devices) or the ability to award large contract orders.

On a related note, the adoption of cloud manufacturing within manufacturing and service industries have led to the rise and adoption of Digital Thread (DT) centric manufacturing system architectures. The DT spans across multiple stakeholders (machines and organization) within the folds of a cloud manufacturing based digital services. The absolute adoption of the architecture is contingent on similar factors as mentioned by Oliveira et al. [16]. The collection and transfer of digitized manufacturing data across the thread has opened business, legal and technology concerns. Manufacturing organizations have always shied away from the sharing of sensitive or proprietary manufacturing data across cloud-based platforms. A key challenge of such platforms connected via networked architectures is the concern over cybersecurity and malicious intrusions that intend to fish for or modify proprietary data across the DT [19]. Various entities across the thread have disparate software, technological expertise, and infrastructure stacks. This leads to the design and implementation of incompatible middleware architectures through which these organizations interact with the DT. This in totality contributes to a lack of standardization in how manufacturing entities today interact through the DT and share information. Another issue of concern that arises is the ownership of the data and the central infrastructure that connects the different entities and stakeholders. Service providers will lose control of the data generated and shared with the intermediary agent and consequently with the client.

In summary, pervasive adoption of CMaaS is hindered by the added complexity and ensuing technological unreadiness of the end users. Additionally, the allied DT infrastructures required to keep such platforms operational are often sources of concern to manufacturing entities due to privacy, data security and data proprietorship issues. Motivated by these points, this paper aims to answer the following research questions: (i) How can CMaaS platforms be made more accessible to end users through considerable shifting of technological and intellectual overheads away from the users (ii) How can the power of distributed ledgers be harnessed to address data security and privacy issues in DTs (iii) How to appropriately model manufacturing assets on distributed manufacturing systems (iv) How to model information exchange and interactions between different parties on such systems and finally (v) How to achieve the aforementioned goals without introducing extremely disruptive changes that could otherwise inhibit CMaaS adoption.

This paper describes the design and evaluation of system architectures for CMaaS platforms through a cyber-manufacturing software middleware stack with a decentralized system enabling the DT infrastructure connecting data associated with the movement of physical parts across manufacturing partners (Fig. 1). Besides conventional upload of parts by clients to a cloud manufacturing resource, this work demonstrates how job shops themselves can share data assets as a manufacturing resource to be parametrically configured based on the clients' needs. The contribution of this paper is to propose middleware architecture solutions for CMaaS platforms that would reduce the technological and intellectual burden on the end user and at the same time make use of extant data assets to make the best use of manufacturing resources. The proposed architecture also aims to demonstrate how decentralized architectures enabled by the Blockchain technology can be used to benefit clients and corresponding manufacturing service providers (e.g., job shops) through retaining data ownership and control over how much data is shared across the distributed ledger. This paper is organized as follows. In section 2, the related works, status quo and state of the art for CMaaS platforms and Distributed Ledger Technology (DLT) based manufacturing systems architectures are described. In the subsequent part of section 3, focus is primarily shifted towards the design of the various middleware in the proposed decentralized CMaaS platform and the description of a standardized data model for manufacturing assets on DLT platforms by modelling their digital twins [20] as non-fungible assets on the Ethereum blockchain framework. Section 4 describes the implementation of the various middleware and the evaluation of the digital manufacturing system as ERC-721 non-fungible tokens (smart contracts) on the Ethereum blockchain by demonstrating their performance on three Ethereum based test networks [21].

2. Related work

Contemporary CMaaS platforms can now take advantage of digital twins of manufacturing assets made possible by the evolution of improved virtualization technology and a host of other enabling technologies and systems [22]. Cloud Manufacturing and Industry 4.0 revolution along with the concept of digital twins have now led to manufacturing paradigms with increased product data proliferation along a digitized manufacturing system with increasing degrees of end user interaction. This has allowed the spawning of a multitude of CMaaS platforms in the recent past wherein the digital twins of products on the cloud serve as accurate surrogates of physical assets. In the recent past, additive manufacturing based CMaaS platforms have seen a sharp rise in the implementation and exploitation of digital twin-based product life cycle management. These platforms have shown to efficiently exploit on-demand access to a shared collection of distributed manufacturing assets to form reconfigurable production lines which lead to higher process efficiency, reduced product costs, and optimal resource allocation [23].

Cloud based platforms like Shapeways [24] originated from the idea of building a 3D printing marketplace and community. The success of Shapeways has inspired many other startups and fully-fledged cloud manufacturing companies to start offering services spanning from polymer based additive manufacturing to CNC machining. With the evolution of high performance cloud-computing resources giving access to accelerated compute-intensive tasks [25], companies like CloudNC [26], Plethora [27], Fictiv [28], and Xometry [29] have evolved and now they host CMaaS platforms promising to make mass manufacturing more accessible to end users through the design of proprietary cloud enabled software platforms. Cloud based manufacturing platforms like Quickparts [30] and LiveSource [31] in addition to the previously mentioned CMaaS platforms allow end users to upload their CAD data from a variety of proprietary software packages. Contingent on subsequent geometric validation of these CAD models, these platforms instantly suggest and propose lists of qualified manufacturers who can manufacture these digital models. These platforms now enable product designers and engineers to instantly access the capacity of a network of manufacturing facilities following a model aptly called the "Uber of Manufacturing Services". Contemporary research in these avenues now provide mechanisms which allow accurate mapping of these distributed manufacturing resources and remote dynamic service requests typical in these large scale CMaaS platforms [32]. The instant quoting engines hosted by these platforms, now at the click of a button, can evaluate the manufacturability of products and return to the user RFQ replies.

Despite the contributions of new and emerging CMaaS platforms in the cloud manufacturing landscape, contemporary platforms suffer from significant limitations in that the role of the intermediary agent connecting the client with service providers can severely limit the amount and type of transactions carried out. The centralization of data within an intermediary agent can lead to issues of data ownership, integrity, and liability concerns. To add to the list of limitations, current CMaaS platforms do not meet the major requirements – the need for an open source software framework that supports data intensive,

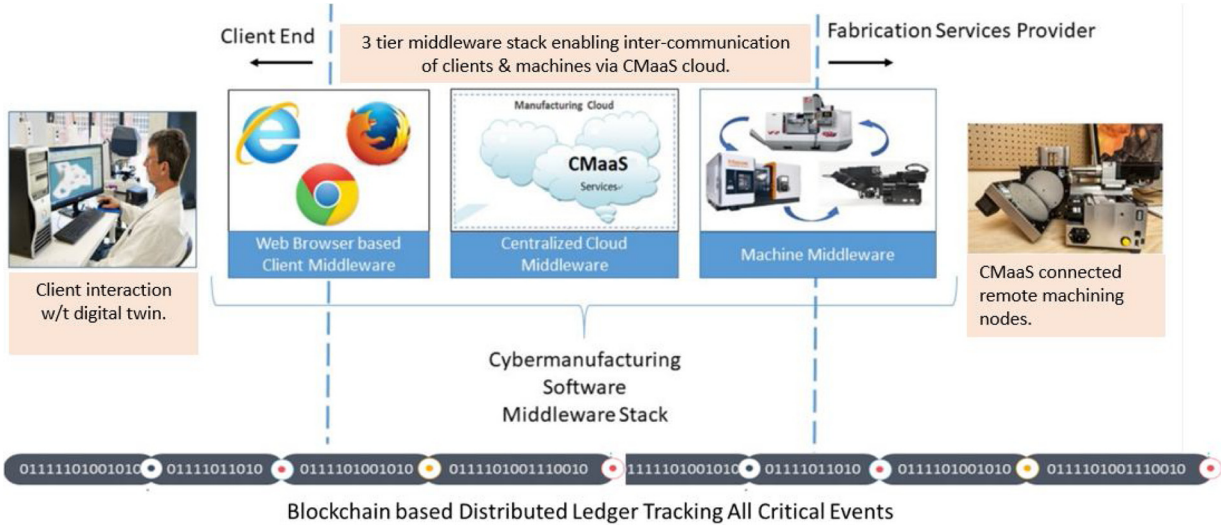

Fig. 1. Cybermanufacturing Middleware Software Stack with underlying decentralized data ledger enabled by Blockchain technologies recording critical events.

distributed applications, and – the need for cloud based, multi-tenancy software-as-a-service (SaaS) architectures as described by Wu et al. [23]. Many CMaaS platforms do provide access to end users shared software applications (e.g., ERP, CAD/CAM) under the envelope of SaaS [33,34]. However, the assumption that many of these CMaaS platforms make about the technical and resource capacity of the end user is often times over estimated. It sometimes is just not pragmatic to think that a non-expert end user would have instant access to and skill of CAD softwares to bring about rapid parametric changes to a digital model and be able to re-upload the model on design changes. There has been past research wherein attempts have been made to find out how cloud manufacturing platforms could help non-expert customers use services more efficiently. Wang et al. [35] have investigated how Internet of Things (IoT) enabled cloud based additive manufacturing platforms can be used to provide customers sufficient information and support throughout the entire product development process. However, most of the existing literature tender solutions that involve relatively large-scale hardware and software-based modifications which can be considered disruptive by manufacturing service providers. The contribution of this paper in this aspect is to design and implement "plug-and-play" middleware software architectures which allow end users, CMaaS platforms and manufacturing nodes to interact in seamless and coherent fashions. The aim is to achieve these goals in such a way that a significant overhead is shifted away from the end user and at the same time no significant burden is imposed to any other entity as an upshot, thereby contributing to an overall improvement of cloud manufacturing democratization and adoption.

The concept of DTs in cloud-enabled smart manufacturing is not new. The National Institute of Standards & Technology (NIST) estimates that manufacturing infrastructures empowered by CPS and DT would save the U.S manufacturing around $100 Billion annually [36]. Emboldened by that motivation, the NIST through its "Digital Thread for Smart Manufacturing" project has been in active development of methods, protocols and standards for DT running through design, manufacturing and support processes [37]. Subsequent research and development of enabling technologies for DT by the NIST has been multifaceted. The outcomes of these investigations have manifested as the design of CPS based product lifecycle test beds by Helu et al. [38], feasibility analysis of STEP AP242 standards for manufacturing information exchange by Fischer and Trainer et al. [39,40], the use of digital certificates for data traceability by Hedberg et al. [41], the use of Quality Information Framework (QIF) standards for metrology data exchange by Michaloski et al. [42] the application of model-based manufacturing by Hedberg et al. [43] and the architecture design for data management of connected systems by Helu et al. [44] to name a few. There have been significant strides in the research of product data models and architectures for DTs as well. Specific to Additive Manufacturing (A.M), Bonnard et al. [45] have introduced the hierarchical object-oriented model for digital chains for better encapsulation of closed-loop manufacturing data. Lu et al. [46] have proposed a service-oriented product data model for cloud manufacturing that can effectively capture changing requirements on downstream production activities. Gan et al. [47] have shown mathematical models for product manufacturing information flow that encapsulates increased customer involvement and feedback typical in cloud manufacturing DTs.

To address the issues of product data privacy, security, ownership and traceability on DT's for manufacturing, numerous initiatives have been taken in the recent past. A technology that has come to the forefront as a promising solution is the Blockchain based DLTs. According to NIST, blockchains for smart manufacturing can not only provide tamper-proof transmission of manufacturing data, they also allow seamless traceability of the data to all participants in the production process [48]. Major research thrust areas in blockchain based DLT platforms have been related to investigations of product data models on DLTs and architectural consideration for CMaaS infrastructures that ought to operate on such platforms. Lee et al. [49] have proposed a unified, three level blockchain architecture for highly connected Cyber-Physical Production systems (CPPS) that attempts to act as a guideline for the implementation of DLT in manufacturing paradigms. Madhwal et al. [50] introduces the potential application of DLT in managing supply chain for aircraft spare parts that come with hundreds of parametric complexities. The impetus is to achieve a transparent network of supply chain for the aircrafts' parts which eventually reduces the possibility of the availability of the parts in the black market. In a case study of blockchain technology in manufacturing, the authors Angrish et al. [51] have proposed and implemented a blockchain-based, decentralized model for handling manufacturing information generated by various stakeholders in a supply chain network. The authors have introduced the "FabRec" model which houses a decentralized consortium of manufacturing machines and computer nodes autonomously acting and negotiating through computer-coded smart contracts to enable organizational transparency and provenance through a verifiable audit trail. Another interesting implementation of blockchain manufacturing system solution in the paradigm of cloud manufacturing and CMaaS platforms was found in the research of Li et al. [52]. Their research has proposed a Blockchain Cloud Manufacturing "BCmfg" architecture to facilitate the development of a distributed, peer to peer network of cloud manufacturing nodes. The authors showcase an

implementation case study wherein a MATLAB Simulink based virtualized digital twin of machine tools is allowed to interact over a Multichain [53] based blockchain network with other verification and participatory nodes. In subsequent research related to integration of blockchain technology in A.M and CPS, Mandolla et al. [54] have explored the design of digital twins for A.M through blockchain exploitation in the context of highly regulated aircraft industry. Huang et al. [55] have proposed blockchain based data management of digital twins for complex interconnected parts. Barenji et al. [56] have shown impressive work into solving scalability, security, data ownership and big-data problems typical in CPS enabled cloud manufacturing platforms through the implementation of blockchain-based platforms to eliminate third-party problems found in centralized systems. Lemeš et al. [57] in their work have demonstrated how distributed CAD environments enabled by blockchains can be exploited for secure manufacturing collaboration.

A comprehensive investigation of the past work in the field of DLT based manufacturing system architectures for CMaaS platforms mentioned hitherto have shown many promising architectural frameworks and data models that attempt to solve the aforementioned data security, ownership, privacy and trust issues often faced in centralized systems. However, there are some gaps in the existing research that this paper attempts to propose solutions for. The limiting gaps identified are as follows: (i) Very few of the extant research have taken steps to describe how the various stakeholders interact over a decentralized architecture. These interactions are non-trivial and require studying how complex computer code based smart contracts can replace some of the manual, labor intensive work carried out between tiered partner organizations. Useful guidelines have been proposed but most lack key implementation details which limits the community's understanding on whether decentralized manufacturing systems enabled by Blockchain technology can truly be beneficial in an Industry 4.0 environment and beyond. (ii) There have been several proposed product data models of assets for DT based architectures. However, there still seems to be lack of consensus as to what should be an appropriate data model for assets on blockchain based DTs. Many of the contemporary research have proposed blockchain data models directly stored on the distributed ledgers. However, blockchains frameworks are not ideal as storage solutions. (iii) CMaaS platforms based on blockchains should consider, in addition to product data models, appropriate data models for fungible asset transfer in order to truly automate contracts between end users and service providers. There seems to be very little work done to propose how a fungible asset transfer should be occurring alongside a manufacturing asset on DLTs. (iv) Many of the existing research showcase applications based on permissioned consortium blockchain solutions or blockchain networks simply deployed on local nodes. These deployments do not give a comprehensive picture as to how the proposed models would behave in true public blockchain networks where there could be millions of transactions happening in a short period of time as would be expected in large blockchain based CMaaS platforms. (v) Case studies considered in existing research often fail to consider very important security practices both in smart contract code and in overall blockchain framework deployment (e.g., storage of sensitive cryptographic keys in local nodes as opposed to using identity management software). There often seems to be the absence of the design of middleware that would allow the proposed solutions to seamlessly function with existing industry grade and secure authentication and identity management solutions for blockchain interaction. (vi) Many of the existing research showcase programmatic blockchain solutions implemented through very high-level languages like JavaScript with dynamic types. While such implementations would be appropriate for a wide range of applications, for blockchain based CMaaS platforms also handling fungible assets, that is often a risky choice. For example, the Bitcoin scripting system is purposefully not Turing complete [58] to ensure any fungible asset transfer occurs in a completely deterministic fashion.

This paper addresses some of the aforementioned knowledge gaps in relation to building a decentralized manufacturing system particularly in the context of cloud-manufacturing services delivered over a publicly accessible internet. In this context, the contribution of this paper is: (i) To propose data models and design patterns through object-oriented paradigms to represent manufacturing assets' digital twins and information exchange via blockchain smart contract architectures (ii) To demonstrate how globally accessible public blockchain networks with their secure fungible asset transfer models can be exploited for automated contract negotiations in CMaaS platforms (iii) To demonstrate the efficient design of smart contract code constructs that allow for appropriate representation of the data models while satisfying all the restrictive security features of proven blockchain programming languages like Solidity [59]. (iv) To propose important performance metrics that the community can utilize to assess the efficiency of their own smart contract code constructs and finally (v) To design and implement middleware software architectures that allow the blockchain interface to seamlessly interact with the CMaaS platform without introducing significant disruption.

3. Cloud Manufacturing-as-a-Service middleware service architecture

3.1. System architecture roles, functions and data flow

A high-level overview of the proposed system is exhibited in Fig. 2 explained in the context of a design and servicing of a CNC machined part. It can be started with the assumption that models of parametrically configurable, consumer or industrial parts are hosted by manufacturers on the CMaaS platform and clients want to fabricate versions of those parts with dimensions that suit their respective needs. The system can be differentiated into 2 subsystems, namely the upper Physical Manufacturing system layer and the lower Blockchain Network Layer. The process starts with a client or end user modifying a parametric model of the digital twin of a product model on a web browser through a front-end app hosted by the CMaaS platform. Should the client be satisfied with the final form of the rendered digital twin, a purchase order or an RFQ command can be initiated. The client middleware subsequently makes Remote Procedural Calls (RPC calls) to functions encoded within a manufacturing system smart contract hosted on a blockchain network. The manufacturing system smart contract, on receiving metadata about the part to be manufactured, instantly returns a quotation through blockchain events communicated from the blockchain layer to the physical manufacturing system layer. If the client approves this quote, the client can initiate and authorize a make order through the middleware against a purchase order id sent to the client in the previous step. This process initiates a blockchain event to which the client middleware subscribes to. On reception of the make order event from the blockchain, the client middleware subsequently sends off toolpath regeneration commands to the CNC toolpath regeneration engine hosted by the CMaaS platform. Another session of client and CMaaS middleware communication eventually leads to regenerated toolpath that corresponds to the parametrically altered model required by the client. On successful production of a part, the blockchain network layer completely takes over. The termination of the machining process is registered by the CNC machine on the blockchain as an event and this leads to the creation of an ERC-721 [60] modelled non fungible token representation of the part on the blockchain with the initial ownership assigned to the CNC machine owner. The blockchain network also takes care of the automatic payment from the client to the CMaaS platform and autonomously negotiates refunds and returns through self-executing smart contract codes running on the blockchain and catering to the smooth operation of the manufacturing system.

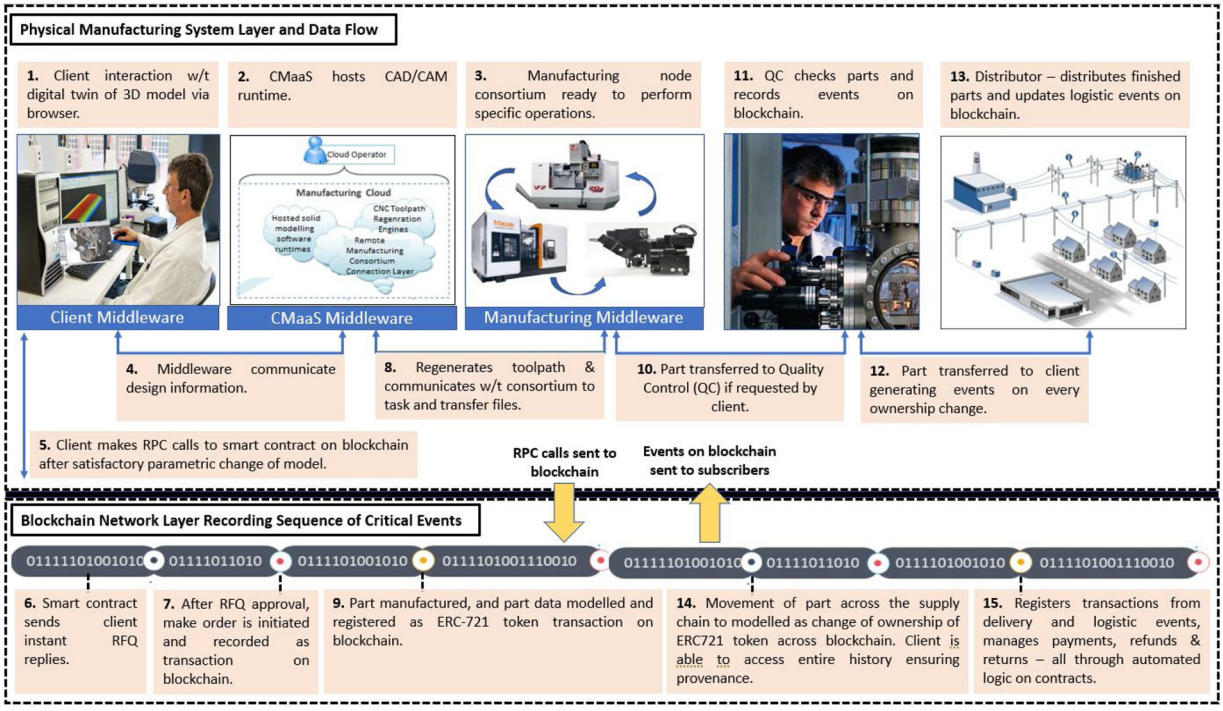

Fig. 2. Holistic overview of the implemented decentralized CMaaS platform architecture.

3.2. CMaaS service architecture

Fig. 3 shows the architectural elements and the salient features of the overall CMaaS middleware. One major consideration into the design of the architecture was to enable the CMaaS middleware to act as a plug-and-play interaction layer capable of interfacing with any CAD runtime hosted anywhere. Most conventional CAD software packages give access to users an Application Programming Interface (API) [61] (Section II in Fig. 3). To deliver data and functionality across multiple remote clients, add-ins were designed to spawn multiple threads from within the single process of the CAD runtime engine in a cooperative multi-tasking paradigm. The child thread and the main thread API interface communicates through a Publisher/Subscriber (P/S) layer (section III Fig. 3) wherein the child thread registers specific events asking for permission to gain access to API functionalities. The API engine in turn can subscribe to these events through the P/S layer so that it can respond to the requests coming from the child thread. Subsequently, the API, through its own implemented functions, makes parametric changes to the model housed in the UI of the CAD runtime. On successful change to the parameters of the model, the API automatically collects new metadata pertaining to the parameters of the altered model. The API in turn replies back to the web server thread

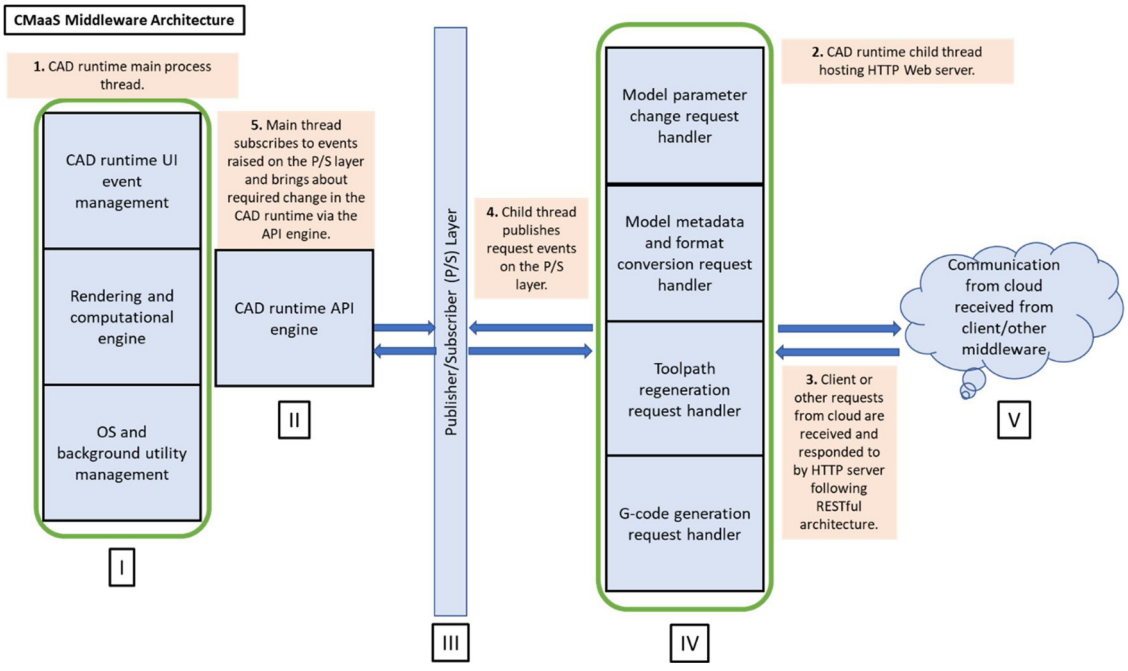

Fig. 3. Architectural elements of the CMaaS middleware.

with this new model metadata. The server thread receives this data and relays it back to the user's parameter change request. This metadata is eventually used by the client middleware to bring about parametric changes to the digital twin being rendered on the front-end app, thereby completing the request-response cycle.

3.3. Client (design) services middleware

Present manufacturing paradigms in the settings of consumer and industrial production, frequently involve rapid parametric changes of computer-generated models that products are based on. These parametric changes are mandated to cater to different consumer requirements or industrial applications, and it is for this reason manufacturing entities have devoted a lot of resources and attention to mass customization of products to satisfy product family and product platform requirements [62]. Contemporary CMaaS platforms assume that the end users have sufficient technological and resource capacities in order to be able to achieve the design change and validation tasks. Such tasks often involve modification of product models in proprietary, computationally intensive CAD software platforms and it is needless to mention that not all end users would necessarily have access to such resources.

Fig. 4 shows the architectural elements of the client middleware designed and implemented in this research. The client middleware establishes and maintains communication with other middleware in the CMaaS platform. Additionally, it communicates with the blockchain network layer and manages blockchain identity through embedded third-party apps. The client middleware allows the client-side software to directly send HTTP requests involving model parameter change, toolpath regeneration requests etc. to the CMaaS platform. This layer, on transfer of the aforementioned requests to the platform, waits for commensurate replies. After the CMaaS platform does the necessary parametric change, model geometry and metadata information are returned back to the client middleware as a response to that request. The client middleware can also support real-time communication and request/response cycle through rendering of a machine coordinate axis rendering engine for verification and monitoring purposes.

Section IV of Fig. 4 shows the blockchain identity and wallet management layer of the client middleware. This layer acts as the interaction layer between the client front-end app and any third party blockchain wallet app [63] that allows the client user to prove his/her identity on the blockchain. The client interacts with a digital twin of a model on the front end and brings about parametric changes to the twin to suit his/her needs. When the client invokes a make order to send off the final design for manufacturing, transactions are generated on the blockchain to record this event so as to keep a continuous trail of immutable information on the chain corresponding to the evolution of the product being manufactured. The registration of transaction on the blockchain requires the client sign the transaction messages with a private key to prove blockchain identity and ownership of the message within the folds of the digital signature algorithms. Section IV of the client middleware has the responsibility of automatically initiating transaction signing process by directly interacting with the third-party wallet apps. The wallet also manages transfer and reception of cryptocurrency (fungible assets) and crypto-tokens owned by the client and the client middleware acts as a media allowing secure transfer of these assets following standard security protocols.

The final section of the client middleware is the blockchain interface layer (Section V Fig. 4). The function of the blockchain interface layer is to enable the client middleware to be able to call functions and attributes, subscribe to blockchain events and generate events on the blockchain as inbound software oracles [64,65]. In order to be able to achieve the aforementioned functionalities, the middleware has to have the ability of making RPC calls to the blockchain network layer. The middleware communicates with third-party APIs that provide RPC call capabilities. Additionally, the middleware has to act as an interaction layer allowing seamless integration of the blockchain wallet app mentioned in the previous step with the blockchain network layer so that message transaction and asset transfer can take place.

3.4. Service provider (machine) middleware

Fig. 5 shows the architectural elements of the manufacturing middleware. The manufacturing middleware initiates by spawning a thread on the machine OS that runs concurrently to the main thread controlling the machine motion commands. The child thread spawned also starts a multithreaded HTTP web server within it and it is this server that establishes an inter-server communication with the CMaaS platform and responds to asynchronous calls from the client middleware. The client middleware interaction layer is substantiated by server

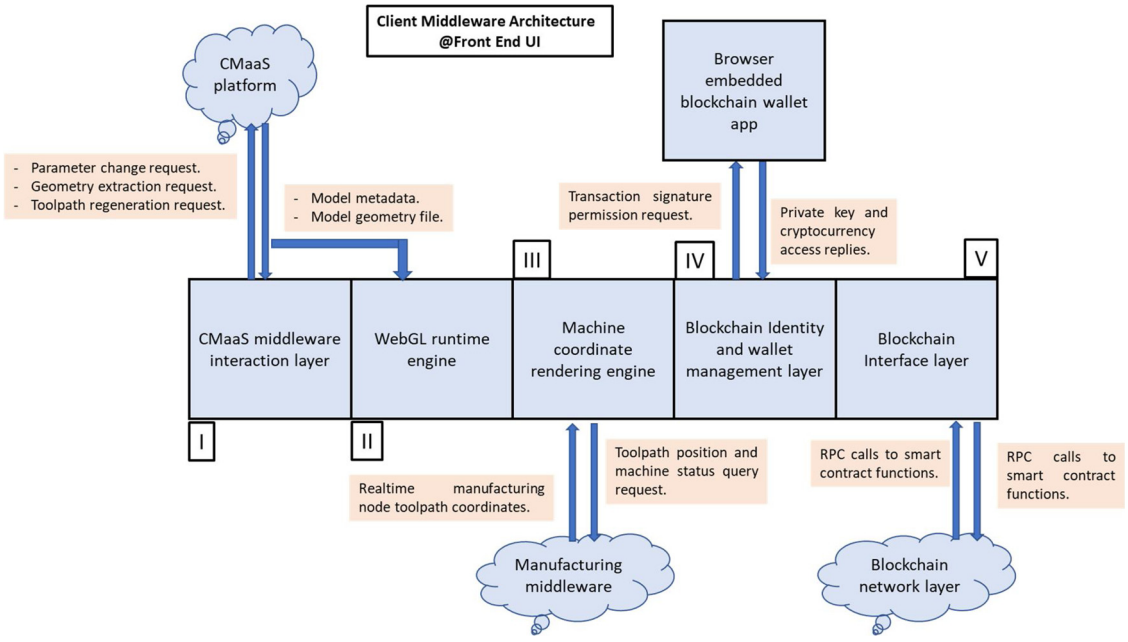

Fig. 4. Architectural elements of the Client middleware at the front-end UI presented to client user of the CMaaS platform.

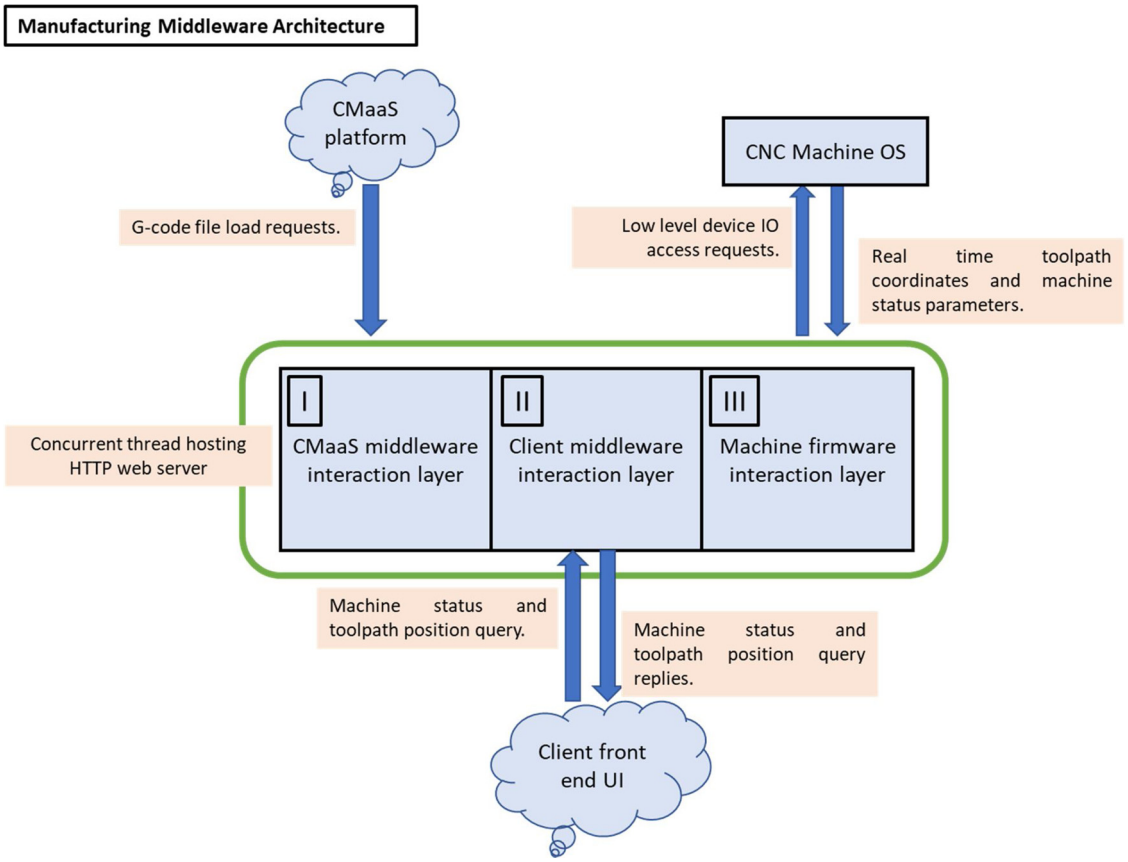

Fig. 5. Architectural elements of the Manufacturing middleware.

endpoints that cater to requests coming from the client middleware. The machine firmware interaction layer (section III Fig. 5) is the backbone behind the query and access to low level machine status parameters. Machine input/output (IO) statuses, process parameter statuses and real time toolpath coordinate positions are all low-level hardware-based parameters that can be queried through the machine firmware interface. Depending on the manufacturer of the machine, an API can expose high level functions and attributes that give access to these low-level firmware parameters allowing seamless integration with the server platform of the middleware.

3.5. Decentralized part tracking and digital assets provenance through smart contracts

A CMaaS compliant blockchain manufacturing system network was designed and implemented as a part of this research. The rise of the DLT and blockchain based enterprise solutions has seen a sudden rise in the application of blockchain in the manufacturing system and logistics industry. Blockchain based manufacturing system networks have been employed to track product history in agriculture [66], to ensure provenance in the diamond industry [67], to enable traceability for food safety [68] etc. Truly decentralized blockchain solutions are still limited and in addition to the technical and resource overhead requirements for expertise in the field, blockchain manufacturing system frameworks can often appear obscure and difficult to adapt to. Consequently, many blockchain based manufacturing system solutions lack standardization, and do not follow industry best practices which directly affects the manufacturing system efficiency. This work proposes a systematic way of designing blockchain based manufacturing system networks for CMaaS platforms by giving special attention to smart contract features, methods, attributes and events.

Separation of Concerns in Smart Contract Structure: Separation of concerns [69] is a software engineering concept wherein the core algorithm of a software engineering project is formally separated from special purpose concerns such as synchronization, real-time constraints, or location control. A three folded separation of concerns in a manufacturing system smart contract implementation is demonstrated in this research. A typical CMaaS manufacturing system involves 3 major elements: (i) The stakeholders, (ii) The Asset and (iii) The Core manufacturing system network. The stakeholders are the participants who interact with the manufacturing system. This stakeholder thus encompasses the client or customers, the CMaaS platform, the manufacturing nodes (CNC machine owner), the raw material collectors, the quality control personnel, the verifiers, the distributors and so on. The asset represents the product moving through the manufacturing system – from the manufacturer to the consumer. All of the participants in the CMaaS manufacturing system must have different levels of permissions and authority over the manufacturing system network. For example, a registered CNC machine owner cannot also act as a distributor with his/her blockchain identity on the manufacturing system. Similarly, a client cannot be allowed to alter administrative attributes of a deployed manufacturing system smart contract. Only the smart contract deployer can be allowed to perform that task whether that be the CMaaS platform or the CNC machine owner. The roles, permissions and code construct defining the stakeholders should not be a part of the code construct of the manufacturing system smart contract or the asset smart contract. This leads to a three-fold separation of concern paradigm where three different smart contract architectures control the behavior and the attributes of the stakeholders, assets and core manufacturing system network.

Object Oriented Design Patterns and Inheritance Structure: With separation of concerns comes an equally important yet heavily overlooked design pattern in smart contract software engineering paradigms. This includes the application of object-oriented design

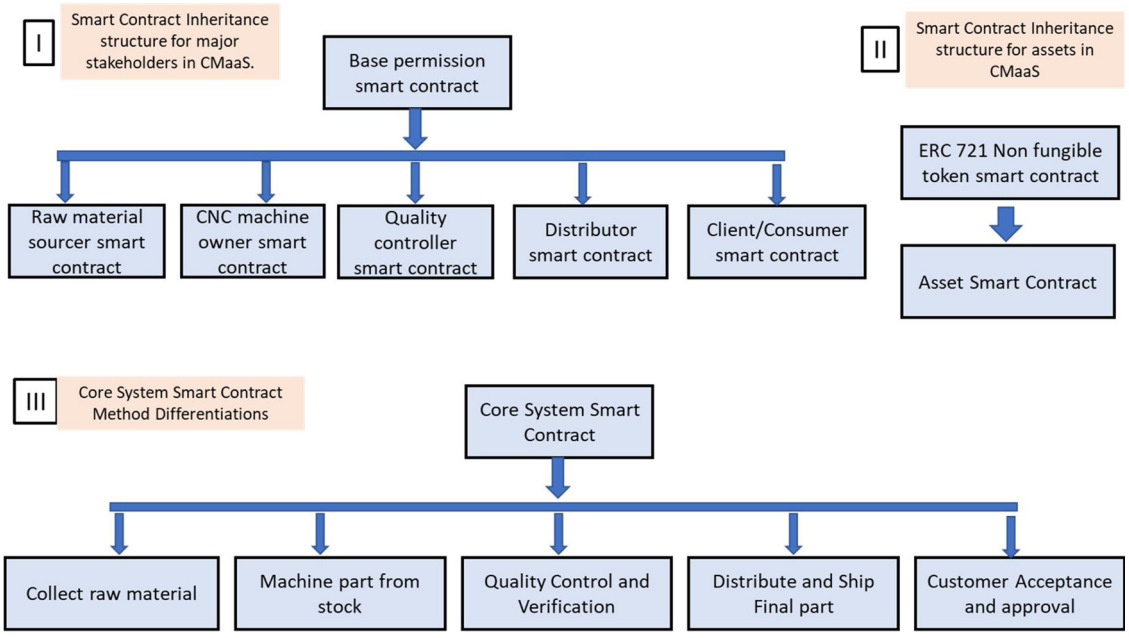

Fig. 6. Smart contract inheritance structure of major CMaaS manufacturing system elements.

Table 1
Typical methods and attributes in a base permission smart contract.

Methods	**addMember** Input: (member role, account address)	Adds members to the struct Role
	removeMember Input: (member role, account address)	Removes members from the struct Role
	alreadyMember Input: (member role, account address) Returns: Boolean membership status	Checks membership of an address in the struct Role
Attributes	**struct Role** {mapping (account address = > Boolean membership status) members.}	A struct data structure mapping addresses to Boolean statuses of membership

patterns [70] in smart contract code constructs. Fig. 6 shows the object-oriented smart contract inheritance structure of the major CMaaS manufacturing system elements. Section I of Fig. 6 shows the inheritance structure of the manufacturing system stakeholders. Instead of creating completely disparate smart contract code constructs for each of the stakeholders, they can inherit methods, behaviors and attributes from a parent smart contract. This smart contract is shown as the base permission smart contract in the figure. The base permission smart contract is a generalized smart contract code construct which controls basic permission protocols allowing the stakeholder members to interact with the manufacturing system. Additional methods and attributes go into the code constructs of the inheriting smart contracts that provide them additional nuances. For example, the base permission contract can contain method implementations to add members, remove members and check membership validity. A sample version of the base smart contract with its methods and attributes is shown Table 1. This inheritance model allows the client smart contract to register new clients, remove existing clients from registration or check registration status of clients on the blockchain manufacturing system. The Role struct shown in Table 1 can be inherited by the client child contract as a library member and polymorphed into specific role of clients. In addition to these methods and attributes, the client smart contract can augment itself through the addition of methods and attributes specific to the client. This process is true for all the stakeholders with each inheriting smart contract containing additional methods and attributes specific to the stakeholder type.

Section II of Fig. 6 shows the inheritance structure of the asset smart contract. The asset represents the data model of the part being manufactured and moved through the blockchain manufacturing system. Since the implementation in this paper deals with the Ethereum blockchain, the asset can be modelled after the ERC-721 non-fungible token standard [60]. Manufactured products vary in value, properties, shape, weight, size etc. and hence an ERC-721 standard wherein all tokens are considered to be disparate and unique is an appropriate data model to mimic digital twins of manufactured products on the Ethereum blockchain. The ERC-721 standard defines the minimum interface that ought to be implemented in a smart contract so that the non-fungible token can be owned, managed, transferred and traded. The asset model discussed in this paper is a smart contract which inherits methods and attributes from the ERC-721 standard template. Section III of Fig. 6 shows the core method members of the manufacturing system smart contract proposed for the CMaaS platform. It is to be noted that, depending on the number of stakeholders in the manufacturing system and the nuances in the transition of products through the chain, there can be a multitude of method implementations. We can observe from the block diagram that methods representing collection of raw materials, machining of part from stock, quality control after machining, distribution and shipment of verified part to customer and finally acceptance by customer should be implemented in the manufacturing system smart contract.

Activities and States in a Blockchain based Manufacturing system Framework for CMaaS: The manufacturing system smart contract in this paper implements a set of methods, modifiers, attributes and events, a simplified version of which can be seen from the state diagram of Fig. 7. In the top layer of the diagram, the stakeholders who are involved with the ownership of the part at a particular instance of time are shown. The part starts as a raw material under the ownership of the raw material sourcer. The third layer from top labelled as the

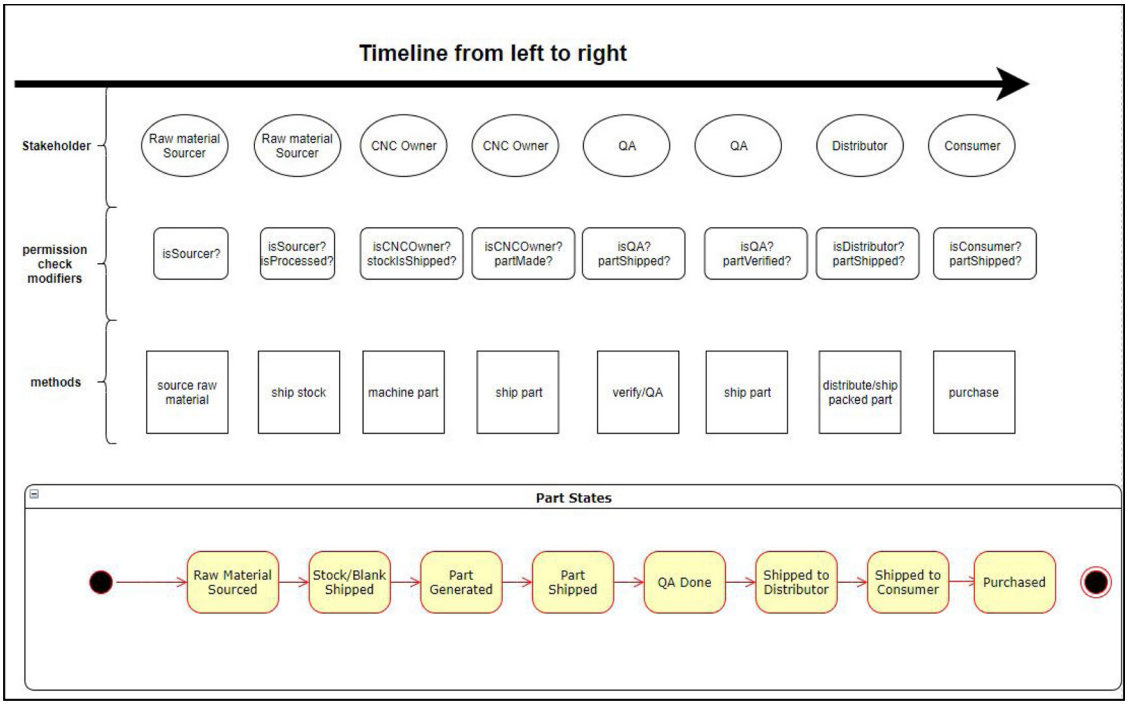

Fig. 7. State diagram showing major methods and events on the manufacturing system smart contract.

"methods" layer shows the major method implementations in the manufacturing system smart contract. For example, an entity that sources raw material in the stakeholder layer would invoke the "source raw material" method in the method layer. The invocation of this method will mint a new non fungible ERC-721 token on the blockchain and associate it with the generation of the part as a raw material. This generates an event in the blockchain represented by the "Raw Material Sourced" state of the part in the bottom most layer. When the raw material is shipped to the CNC owner, the "ship stock" method is invoked and this emits another event, changing the state of the part to "Stock Shipped" state. In this manner, as the part traverses the entities within the manufacturing system network, the state of the part changes, corresponding events are emitted, and different methods are called. The event emitted can be subscribed to by different stakeholders depending on their interest. The state diagram in Fig. 7 also shows an intermediate layer labelled as the "permission check modifiers" layer. This layer ensures that a stakeholder has the necessary permission rights to invoke a function from the methods layer. For example, when the distributor has to ship the final part to the client, the "distribute/ship packed part" method from the methods layer has to be invoked. From Fig. 7, it can be observed that there are two permission check modifiers namely "isDistributor?" and "partShipped?". These modifiers first check if the stakeholder invoking the method is in fact a registered distributor on the blockchain network and subsequently whether the part to be distributed has passed through the previous state of shipment from the verifier to the distributor. If these permission checks are passed, the distributor is allowed to continue with invoking the function. Questions might arise as to how a modifier in the smart contract of the manufacturing system is being able to call a method residing in another smart contract. This is achieved via inter-contract data exchange on the Ethereum ecosystem and is depicted through Fig. 8.

The Ethereum ecosystem is governed by the Ethereum Virtual Machine (EVM) [71] which is like an operating system establishing communication across all nodes in the Ethereum network. The EVM is capable of executing logic, algorithms and processing inputs that are typically found in smart contracts. The EVM defines the "state" of the Ethereum network at a certain point in time. This "state" refers to the states of blockchain network, blocks, mining, nodes, consensus and smart contracts. The EVM has basically three areas where it can store items and act as a low-level database. The first area is the "storage" area which houses all the persistent state variables of a smart contract. Every smart contract has its own storage space in the EVM. Any function call from within a smart contract that leads to a change in these persistent state variables also leads to a change in the state of the EVM and it is these state changing operations which are ensued as transactions and accrue mining fees. The second data storage area is the "memory" area which is used to hold temporary values and function scope variables which are erased between function calls. The third data storage area is the "stack" which can be used to hold small local variables but most importantly the stack trace of function calls within smart contracts.

The ability of the modifiers in the manufacturing system smart contract to call methods and attributes in the stakeholder smart contracts is achieved via inter-contract data exchange enabled by the EVM [72]. Smart contracts that reside in the global scope of the EVM state can communicate with other contracts within the same scope and can establish inter-contract data exchange. This communication is achieved via internal transactions. Internal transactions are like any other transactions on the EVM with the major difference being that they are not generated by externally owned accounts (i.e., an account controlled by a private key under the ownership of a node). Instead they are generated by the smart contracts that are initiating the data exchange process. These internal transactions, unlike transactions ensuing from a state change of the EVM are not serialized. When one contract sends an internal transaction to another contract, the associated code or function that exists on the recipient smart contract is executed by the EVM. Two smart contracts attempting to engage in such exchange can establish communication by referring to each other's smart contract addresses on the Ethereum ecosystem. The manufacturing system smart contract, in this case, has an encoded array of already deployed smart contract addresses of stakeholders that allows it to establish inter-contract data exchange by invoking methods residing in other contracts through address references. This design paradigm is a direct upshot of the separation of concerns principle and objected oriented model of smart contract software engineering. In this way, we can not only keep our manufacturing system smart contract code clean and free of redundant features, we also can reduce space required by the contract bytecode to

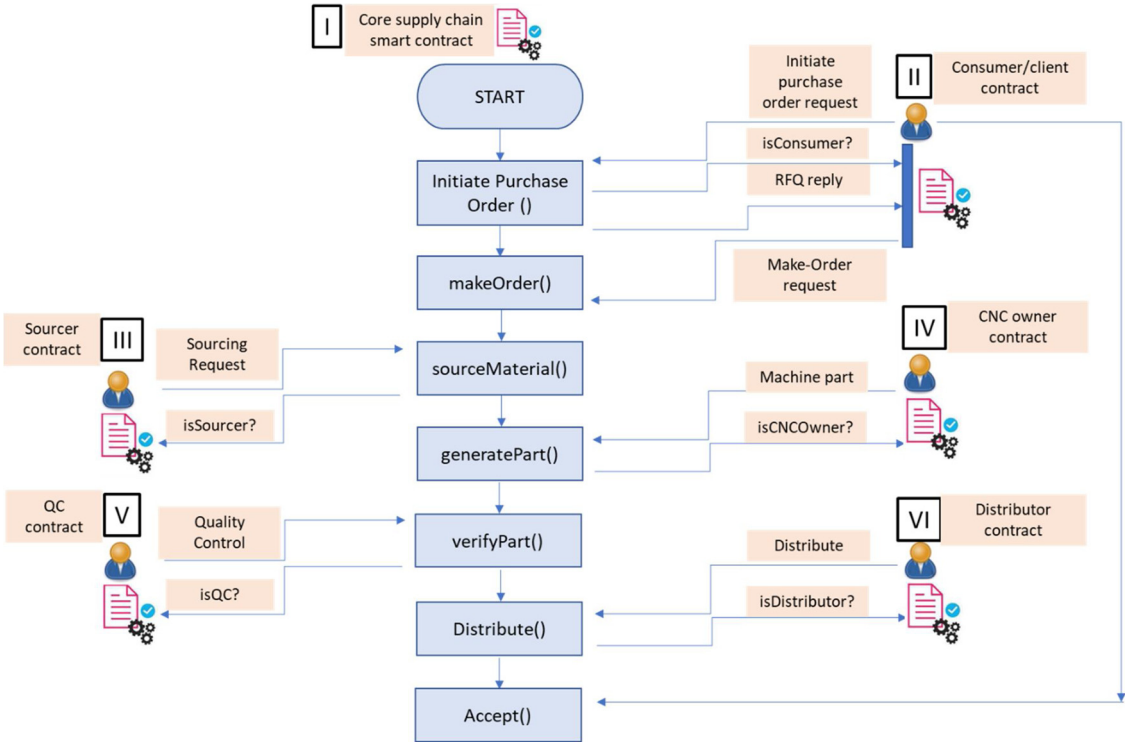

Fig. 8. Inter-contract data exchange between manufacturing system and stakeholder smart contracts.

run on the EVM, thereby drastically increasing runtime efficiency and reducing costs incurred through gas expenditure. 'Gas' is referred to the fee that is to be paid by the method invoker to execute commands on the Ethereum ecosystem [73]. It is a value that indicates consumption towards computational expenses on the blockchain network. From Fig. 8, it can be observed that section I represents the manufacturing system smart contract residing in the Ethereum virtual machine. When a client represented by section II, sends a purchase order request, the corresponding method in the manufacturing system contract is called. The manufacturing system contract in turn, through inter-contract data exchange, makes a query as to whether the method invoker is in-fact a registered client or not by communicating with the client smart contract. If the client is registered, then the manufacturing system contract gets back to the client with RFQ reply. The client can then proceed to requesting for a make order by calling the corresponding function in the manufacturing system contract. In this manner, as the digital asset is passed through the manufacturing system, its state changes, and at every stage, appropriate ownership methods residing within the stakeholder contracts are called from the manufacturing system contract to verify the authentication of the method callers.

4. Implementation and case study

This section will describe the technical implementation of the decentralized manufacturing system for the CMaaS platform. Initially, focus would be given to the technical implementation of the middleware architectures described in Section 3.2 to 3.4. Subsequently focus would shift towards the implementation of the decentralized blockchain manufacturing system network and allied systems described in Section 3.5.

4.1. CMaaS middleware

For this middleware, Autodesk Fusion 360 (Autodesk, CA, USA) was assumed to be running as a CAD runtime engine within the CMaaS server. Multiple clients can interact with 3D parametric models posted by the service provider to the CMaaS server. Utilizing the Fusion 360 API code base, software plugins were written to implement a multi-threaded web server using the Python microweb framework – Flask [74]. The plugins were utilized to spawn servers which followed a RESTful [75] architecture. Consequently, end points were coded which were function handlers capable of responding to remote client requests. Since the web server child thread ran within the same process of the CAD runtime, it has access to all the API functionalities hosted by Fusion 360. The back end of the server thus contained codes that allowed the server to interact with the API and bring about changes in the CAD environment. For example, when a parameter change request comes from the client middleware, an endpoint in the Flask server backend handles this request and calls the required functions in the API to directly bring about parametric changes to the model housed within the CAD runtime. It also handles updates of the 3D CAD model to reflect new dimensional changes made by the client user. Back and forth data transfer and data payload between various middleware (as described below) are handled using HTTP protocols and RESTful design architectures.

4.2. Client interaction with CMaaS middleware

The client middleware controls communication of client requests with the CMaaS platform, manages rendering of models on the client front-end and manages blockchain identity and interaction with the blockchain network layer. The CMaaS middleware interaction layer of the client middleware was designed as an asynchronous call scheduler within a JavaScript runtime engine controlling the front-end client app (Fig. 9a). This scheduler was able to make asynchronous GET and POST requests to the CMaaS middleware whenever the client had to initiate parameter change or toolpath regeneration commands. The client middleware also renders the digital twin of the model being hosted in the CAD platform. The model metadata and the STL geometry file of the models were used in unison to render a model for the digital twin through WebGL technology encapsulated within a high-level wrapper library of Three.js [76]. The client middleware also houses a machine

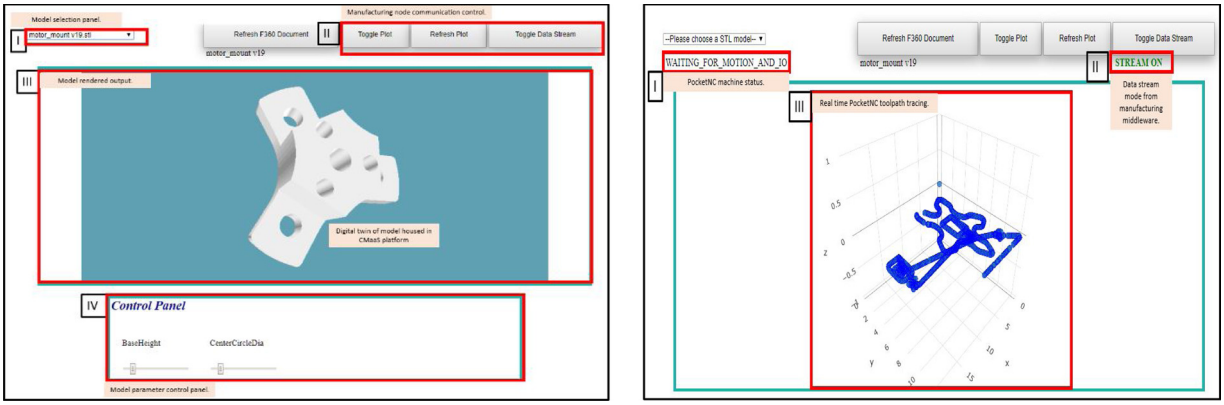

Fig. 9. (a): Client front-end app; Fig. 9(b): Client front-end app with real time rendering of CNC machine toolpath coordinates.

coordinate rendering engine. It makes asynchronous calls to query the real time coordinates of the CNC machine delegated to machine the model of the client (Fig. 9b). Finally, the client middleware function included blockchain identity and wallet management and interaction with the blockchain network layer. Interface software was written exploiting the functionalities exposed by the web3.js [77] library API. This allowed the middleware to be able to communicate and send RPC calls to remote Ethereum nodes. Additionally, this gave the middleware the ability to initiate blockchain transaction signing process by communicating with a browser embedded Ethereum wallet software (Metamask [78]).

4.3. Service provider (machines) interaction with CMaaS middleware

The manufacturing middleware also involved the deployment of a multithreaded server architecture within the manufacturing node operating system. The manufacturing node or the CNC machine used in this research was a five-axis desktop CNC milling station (PocketNC, MT, USA). The CNC machine hosted a LinuxCNC operating system with a MachineKit interface running within a Beaglebone Black hardware system. The MachineKit OS is capable of running scripting language interpreters and hence in a similar fashion to the CMaaS middleware, an HTTP Flask server was deployed within the OS following RESTful architecture. This server exposed multiple endpoints that were assigned the tasks of responding to CMaaS and client middleware queries. The server also had a backend machine firmware interaction layer wherein the endpoints were allowed to access the machine firmware to query status and process parameter values. This was achieved by interacting with functions from the MachineKit OS python API that allowed the call of high-level functions within the server architecture for low level process parameter queries.

4.4. CMaaS and client interaction with blockchain layer

Various smart contracts were implemented to facilitate automatic interaction between the various middleware components.

Digital Asset Smart Contract: The asset or the digital data model of the manufactured part on the decentralized manufacturing system was modelled after the ERC-721 non fungible token standard of the Ethereum ecosystem using Solidity language. The open source OpenZeppelin Solidity framework [79] was used to inherit ERC-721 interfaces. Table 2 shows some of the important methods, events and attributes implemented in the asset smart contract that inherits an interface from the OpenZeppelin ERC-721 implementation. It can be observed from the attributes field that the asset data model holds information about the price, unique product code (upc), owner address, client information and a history trail of all the manufacturing system stakeholders that the raw material and part has passed through. It can be observed that the contract exposes methods like “balanceOf”, “_mint” or “_burn” which query the number of assets owned by a stakeholder, creates a new asset in the blockchain and burns or destroys the associated data of an asset on the blockchain respectively. When a client of the CMaaS platform initiates the make order to manufacture a part, the order is broadcasted to the blockchain network and the manufacturing system smart contract responds by initiating a request to the raw material sourcer. Once the raw material or the stock of the part is collected, the part starts its life and its data model on the blockchain is generated through the minting of a new asset. As the raw material passes through stakeholders and other entities in the manufacturing system into the final product, the attributes in the struct data model change reflecting the latest state of the physical part on the manufacturing system. Transfer events are generated whenever the part ownership is changed. In addition to the methods implemented in the OpenZeppelin framework, this work proposes the addition of more methods like “putAssetForSale” or “purchaseAsset” shown in Table 2. These methods were implemented to allow the sale and transfer of assets between clients who could use the existing framework of the manufacturing system to establish internal trade and exchange of manufactured parts.

Manufacturing system Smart Contract Implementation: The manufacturing system smart contract implemented as a part of this research included more than 15 event implementations, more than 10 modifier implementations, around 13 method implementations and a multitude of attribute and helper function implementations that work in unison to allow the blockchain based manufacturing system work smoothly. Table 3 shows some of the most important methods in the manufacturing system contract. The first major method implemented in the contract is the “initiatePurchaseOrder” method which is called whenever a client sends a request for quote pertaining to the manufacture of a particular product. This method calculates tentative part price and delivery day depending on the inputs. When this quote’s reply is returned to the client, he/she can decide to proceed with a make order. This corresponds to a call of the “makeOrder” method. The client must transfer cryptocurrency to the smart contract, equivalent to the amount set in the RFQ reply that represents the cost of manufacturing the part. The smart contract manages the currency sent in through this method and stores them in escrow accounts against the address of the clients. This fund is not disbursed to the CMaaS platform or the service provider (CNC owner) until the client receives the part that has passed verification. Once a make order phase is passed, a new part starts its life as the raw material stock. This is when an asset is minted on the blockchain as a data model to correspond to this part. The “sourceMaterial”, “generatePart”, “verifyPart” and “shipPartToTheConsumer” methods are functions that indicate sourcing of raw material, generation of part in the CNC machine, verification of part by the quality control department and shipment of part by the distributor respectively.

Table 2
Typical methods and attributes implemented in an asset smart contract with ERC-721 inheritance. Members marked with * are implemented in OpenZeppelin.

Methods	balanceOf (owner address) Returns: uint256 balance of address.	Queries number of assets owned by a stakeholder.*
	ownerOf (uint256 tokenId) Returns: address of token owner.	Queries the address of owner of an asset. *
	transferFrom (address from, address to, uint256 tokenId)	Initiates asset transfer. *
	_mint (address to, uint256 tokenId)	Mints or creates new assets. *
	_burn (address owner, uint256 tokenId)	Burns or destroys assets. *
	putAssetForSale (uint256 _upc, uint256 _price)	Broadcasts to blockchain network about the sale of an asset – proposed
	purchaseAsset (uint256 _tokenId) public payable	Allows the acquisition of an asset through sale - proposed
Events	Transfer (from, to, upc)	Generates an event when a transfer of asset occurs between stakeholders.*
	Approval (owner, to, upc)	Generates an event on ownership or sale approval of an asset by an owner.*
Attribute	struct **Asset** {uint upc; uint price; address currentOwnerAddress; address sourcerAddress; address payable cncOwnerAddress; address verifierAddress; address distributorAddress; address payable consumerAddress; string consumerName; string consumerLocation; State assetState;}	Data model of an asset on the blockchain encoded as a struct. Struct members dictate the state of an asset at any instance of time – proposed.

As the manufactured part passes through stakeholders in the manufacturing system, these methods are called, events are generated, and the state of the part is changed in accordance to the state diagram in Fig. 7.

Smart Contract Security Considerations: Since manufacturing system smart contracts must deal with fungible assets that possess monetary value, ensuring security within smart contract codes is an important task during the design of the code-based contracts. There are several security concerns that need to be taken care of when coding smart contracts [80]. Those security concerns that are relevant with manufacturing system frameworks are discussed below with concrete implementation guidelines. Fig. 10 shows the implemented manufacturing system smart contract fungible asset transfer model by means of a flow chart. It can be started by assuming that the client has already received a reply for his/her request for quote and is about to make a make order request. The function implemented in the manufacturing system smart contract that handles a make order request is a payable function as can be seen from Table 3. This means that the client must send fungible assets in the form of 'Ether' when invoking this function. This fungible asset must be of an amount equivalent to or greater than the price of the physical non-fungible asset that the client is trying to get made from the CMaaS platform. When the client sends the required amount of Ethers as fungible assets to this method call, the amount is not instantaneously transferred to the CMaaS platform. It is first stored in an escrow account of the smart contract. As the physical asset modelled by the ERC-721 standard passes through all the necessary stakeholders in the manufacturing system, it is finally shipped to the client. In case the client is dissatisfied with the final product, he/she is immediately issued a refund by transferring the amount that was previously stored in the escrow account against the blockchain identity of the client. In case the client is satisfied, the amount is transferred to the CMaaS platform and any overpaid amount is returned to the client. In both cases, after the amount is transferred, the balance in the escrow account against the blockchain identity of the client is zeroed. This seemingly systematic process of dealing with customer payments and refunds in the manufacturing system smart contract can have serious security issues in the form of DAO attacks [81,82].

Since manufacturing system smart contracts deal with the transfer of both fungible and non-fungible assets on the Ethereum blockchain, there should be ways to pause the functionality of a smart contract. Allowing manufacturing system smart contracts to be "pausable" is to prevent financial loss in case any bugs are located within the contract code. For example, should the manufacturing system contract be subject to a malicious DAO attack, then the very least to be done from the side of the manufacturing system administrator is to be able to pause the functionality of the smart contract immediately. This is often referred to as "Stop Lossing" of smart contracts. Smart contract pausability can be implemented by implementing the following steps.

(i) Create a storage Boolean variable only accessible by the manufacturing system contract deployer - pauseStatus variable.
(ii) Add a function modifier to each method in the manufacturing system contract. It can be called the modifier requiresOperational ().
(iii) Create a helper function only accessible by the contract deployer that allows to change the state of the pauseStatus variable. This method can be called setOperatingStatus().
(iv) Implement the requiresOperational() modifier such that it allows the execution of a method (other than the setOperatingStatus method) by an invoker only if the pauseStatus variable is set to false by the contract owner.

Table 3
Important methods implemented in the manufacturing system smart contract.

Methods	1. initiatePurchaseOrder (string memory custName, string memory custLoc, uint volumeClass, uint materialClass) public onlyConsumer(msg.sender)	Returns an RFQ reply with a purchaseOrder code to the client using customer information, asset volume, material classes and hourly rate of manufacturing node.
	2. **makeOrder** (uint purchaseOrder) public payable onlyConsumer(msg.sender)	Sends a make order against a purchase order code from the client to the CMaaS platform.
	3. **sourceMaterial** (uint purchaseOrder) public onlySourcer(msg.sender)	Initiates sourcing of raw material by the sourcer. This is when an asset is minted on the blockchain.
	4. **generatePart** (uint upc) public onlyCncOwner(msg.sender) blankShipped(upc)	Indicates finish of part generation by the CNC machine. Can be called by the manufacturing middleware or the owner.
	5. **verifyPart** (uint upc) public onlyVerifier(msg.sender) partShipped(upc)	Indicates completion of verification by the quality controller.
	6. **shipPartToTheConsumer** (uint upc) public onlyDistributor(msg.sender) shippedToDist(upc)	Called when the distributor has shipped the part to the client.
	7. **receivePart** (uint upc) public onlyConsumer(msg.sender) shippedtoCons(upc)	Called when the client receives the part in the final stage of the manufacturing system.

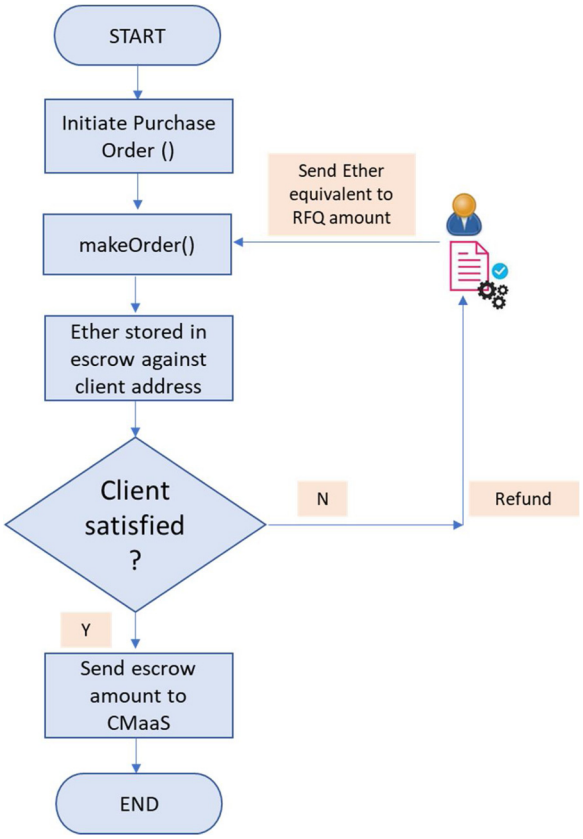

Fig. 10. Manufacturing system smart contract fungible asset transfer model.

Algorithm 1: Multi-Party Consensus Algorithm

Input: M- an integer storage variable, A- array of authorized admins of length N, multiCall- empty dynamic array
Output: Multi-party consensus before function invocation

```
for i ← 1 to N over A do
    if methodCaller == multiCall[i] then
        break;
    else
        append methodCaller to multiCall;
    end
end
if multiCall.length == M then
    invoke admin function;
end
```

Fig. 11. Pseudo Code implementing the multi-party consensus algorithm.

Multi-party Consensus: Blockchain platforms like the Bitcoin support multi-signature accounts that are typically called "multisig". Multisig accounts have the advantage of requiring more than one party to sign a transaction originating from the account. Therefore, they have better protection against theft or unlawful transactions in case bad actors within a smart contract decide to transfer funds elsewhere. For the Ethereum platform, multi-signature protocols can add an additional layer of security. However, multi-sig is not an inherent feature of Ethereum blockchain and hence needs to be encoded in smart contracts. This encoded multi-sign feature in Ethereum is often referred to as "Multi-party Consensus". The basic algorithm that can be employed to implement multi-party consensus in Ethereum smart contracts can be listed down as follows and is shown in Fig. 11 as pseudo-code.

(i) Define an integer storage variable M, that represents the number of keys required to initiate a transaction or invoke a method that changes the state of a smart contract.
(ii) Define an array of administrator addresses of length N, which represents the pool of authorized administrators allowed to participate in multi-party consensus.
(iii) Define an empty dynamic array variable multiCall, that represents the number of authorized administrators which have called a method.
(iv) Inside the method which needs to implement a multi-party consensus, iterate over the array multiCall and check whether the message sender is already a member of that array or not. If yes, it is a duplicate call and return with an error message apprising the method invoker about a duplicate call. If no, append the invoker address to multi-Call.
(v) Next, check if the length of multi-Call is equal to M. If it is, then the method has been invoked for the required amount of times (M times) by M different administrators. When this condition is met, execute the transaction originating from the method call.

5. Results

The manufacturing system smart contract along with the asset and stake holder contracts were all deployed on three different Ethereum blockchain test networks – Rinkeby, Ropsten and Kovan. These test networks vary in the type of consensus algorithms they employ [83]. The Ropsten test network most closely resembles the main Ethereum network since it also makes use of the Proof of Work (PoW) consensus algorithm [84]. The PoW consensus algorithm originally proposed by Bitcoin, is known for its high requirement of computational power. It is a solution to the Byzantine General's Problem on distributed systems [85]. In this algorithm, the network reaches consensus as follows: (i) In PoW, there is an upfront cost of resources called "work" put into generating a block's cryptographic hash value in the blockchain. This work can easily be validated by other nodes in the network to check if it has been done correctly. (ii) Each node in the network is involved in solving a problem (e.g., guessing a random number called the nonce which is a part of the block's hash value) meant to prove that the nodes have done some required work. (iii) Having put in the time to solve the problem is a signal to the network that the result of the work of a node is likely trustworthy and hence a consensus about the state of the network can be reached at. (iv) Nodes trying to solve the problem are called miners and it can take a lot of computer power to solve the problem since the random guessing is done by brute computational capacity. These miners are constantly in a race to solve each problem as quickly as possible and append the next block to the chain. (v) In return of the time and resources invested by the miners, they are paid transaction fees which come directly from the users making the transactions (vi) Once a miner correctly figures out the nonce of a particular block, that block is appended to the blockchain, the miner is paid its fees and the whole cycle resumes for the validation of the next block. This algorithm takes the longest amount of time and fees in verifying transactions. Both the Kovan and the Rinkeby test networks implement the Proof of Authority consensus algorithm (PoA). The PoA algorithm is a modified version of the Proof of Stake algorithm (PoS) [86]. In this algorithm, instead of a node's stake with monetary value as in PoS algorithms, a node's identity performs the role of stake. In PoA, identity is staked by a group of validators that are pre-approved to perform the validation process. PoA like PoS has lower requirement of electric power and the continuity of the network remains contingent on the number of approved genuine nodes. The Kovan test network uses the Aura consensus mechanism which is also known as Authority Round [87].

The manufacturing system smart contract acts as a central hub wherein stakeholders represented by other smart contracts participate and interact. Therefore, testing the performance of the manufacturing system smart contract allowed the indirect validation of the participating smart contracts. The manufacturing system contract was composed of a mix of methods that varied in the number of computational lines of execution. To be precise, methods 1–4 in Table 3 are those

manufacturing system smart contract functions that are involved in data exchange tasks between different smart contracts, new asset generation and minting tasks from inherited smart contracts, blockchain storage variable state change tasks, and so on. Consequently, these four methods were the most computationally intensive and time-consuming functions as far as mining of transactions originating from these method executions are concerned. The other methods were much less resource intensive since they represented either simple transfer of assets between stakeholders or were simple helper functions with few lines of code. Therefore, the gas consumption and validation time performances of transactions originating from these method calls were decided to be evaluated on the three test networks as a measure of the performance of the entire manufacturing system smart contract. The amount of gas consumed by a transaction or a method call over the Ethereum network corresponds to the number of lines of code execution that have to be performed to complete the method call. Higher is this number, larger is the amount of gas fees that has be to submitted by the caller of the function to execute the method within the EVM. Consequently, smart contract code constructs that lead to conservative gas consumption during method calls are generally preferred.

Since the lines of code execution within the aforementioned smart contract methods remained ideally constant, the gas fees used to execute these functions remained pretty much stable throughout the testing phase of the smart contracts on the test networks. However, owing to the different consensus algorithms, the different test networks showed a wide variation in mining times of the transactions ensuing from the method calls. Consequently, it was decided to focus on the mining time distributions for each of the four method calls across the three test networks. To collect gas consumption and transaction mining time metrics, the four methods were called in sequence over the three test networks. Each of these sequences were repeated 40 times to collect a sample for each network that could be assumed to be fairly normally distributed. Therefore, a total of 120 method call transaction data points were generated. Subsequently statistical bootstrapping with repetition was performed on the obtained data to allow for the collection of important centrality and dispersion statistics from the sampling distributions ensuing from the bootstrapping process. The statistical metrics obtained from the bootstrapping simulation are shown in Table 4. Table 4 also shows the mean amount of gas consumptions for each of the methods and the sampling distribution statistics of the mining times of the four smart contract methods across the Rinkeby, Kovan and Ropsten test networks respectively. Table 4 bears testimony to the fact that the mining time for all the method calls are highest across the Ropsten network. This is in accordance to expectation since this is the network that closely resembles the Ethereum main network and employs the PoW algorithm.

The mining process, on average takes the least amount of time on the Kovan network with the Rinkeby network mining performance lying somewhere in between. The mining times of the four method calls within a network were subjected to further statistical testing to find out if there was any statistically significant difference between the means of the mining times within the same test network. For this purpose, Tukey's HSD test [88] was employed following single factor ANOVA analysis on the mining time data. It was found that under a significance level of 0.05, the p-value for the comparison of means was almost zero and hence the null hypothesis that the mean mining times of the four methods within a network are equal was rejected. This means that there is statistically significant difference in the mean mining times of transactions ensuing from the four method calls.

Table 4 also shows the mean gas consumed for the four method call transactions across the different test networks. It can be observed that, for a particular method, the amount of gas consumed due to a transaction ensuing from its call is almost identical across all test networks. The "makeOrder" method had the least mean gas consumption and the "sourceMaterial" method had the highest gas consumption across all the test networks. This was expected since the "sourceMaterial" method initiates the minting of an ERC-721 non fungible token. Therefore, this method in addition to engaging in inter-contract communication with other smart contracts for permission and authorization checks, had to also perform execution in the EVM to establish contact with the asset smart contract. Additionally, in the implementation of this method, there were several contract state change processes associated with the generation of a new asset which might also have contributed to higher gas consumptions. The "makeOrder" method on the other hand involves simple passage of information from the client to the CMaaS platform or the CNC machine owner involving authorization of purchase order and hence did not in general lead to high gas consumption. Judging from the mining time and mean gas consumption metrics, it can be concluded that the manufacturing system and the allied smart contracts would execute most efficiently on test networks like the Kovan test net which employ the Proof of Authority consensus algorithms. The centrality and dispersion statistics associated with the sampled transactions show that the major methods employed within the contracts lead to gas consumption and mining times within reasonable limits of the Ethereum ecosystem.

Since Ethereum smart contracts handle fungible assets and every

Table 4
Centrality and dispersion statistics of major method call transactions over test networks.

				Manufacturing System Contract Major Methods			
				initiatePurchaseOrder	makeOrder	Source Material	generatePart
Test Networks	**Rinkeby**	**Time Stats (sec)**	**Mean**	21.92	16.45	18.04	17.82
			Std. Dev	2.56	2.32	3.19	2.98
			Max	31.66	25.03	27.96	27.14
			Min	13.88	10.63	10.16	6.89
			95% CI	[16.89–26.95]	[11.90 – 20.99]	[11.78 – 24.31]	[11.96 – 23.68]
		Mean Gas Use		196,341	55,212	304,005	94,801
	Kovan	**Time Stats (sec)**	**Mean**	4.07	5.58	4.11	4.13
			Std. Dev	1.87	2.01	1.43	1.78
			Max	12.48	13.25	8.84	12.71
			Min	0.72	0.94	0.89	0.89
			95% CI	[0.42 – 7.74]	[1.63 – 9.53]	[1.31 – 6.92]	[0.64 – 7.61]
		Mean Gas Use		203,611	55,756	305,458	77,214
	Ropsten	**Time Stats (sec)**	**Mean**	28.46	17.07	22.27	21.85
			Std. Dev	7.57	3.51	7.11	4.04
			Max	53.71	27.92	47.30	34.25
			Min	7.44	5.72	4.73	9.01
			95% CI	[13.62 – 43.31]	[10.19 – 23.94]	[8.32 – 36.22]	[13.92 – 29.78]
		Mean Gas Use		203,611	56,168	306,101	74,051

transaction over the EVM incurs gas costs, every attempt should be made to make smart contract code as simplified and less discombobulated as possible so that gas fees are kept at a minimum. High gas fees ensue from higher computational load due to the execution of complex code structures resulting from repetitive branching or multi-level nesting. The adoption of separation of concern principles and object-oriented paradigms provide a structure that allow complexities to be minimized thereby contributing to higher overall smart contract gas consumption efficiency. In a typical CMaaS manufacturing system smart contract, where millions of transactions could occur within seconds, minimizing gas fees bears considerable importance. Therefore, assessment of Ethereum smart contract codes in the form of pre-deployment quality checks for vulnerability and gas consumption efficiency is an indispensable requirement especially due to the immutable nature of the codes post deployment. Hegedűs, P. [89] and Tonelli, R. et al. [90] have previously shown the code performance metrics of Solidity based Ethereum smart contracts using the static Chidamber & Kemerer [91] metrics of object-oriented programming. The authors have presented the software performance metrics of smart contracts across 16 distinct performance metric values over around 40,000 Solidity smart contract source code files through the implementation of their innovated prototype tool SolMet. While the authors focused on a multitude of metrics, only few of the metrics remained relevant to the evaluation of the complexity of the CMaaS smart contract code and those are duly presented in Table 5.

The two most important metric values from Table 5 relevant to the evaluation of complexity of the designed smart contracts are McCabe's Cyclomatic Complexity (McCC) and Nesting Level (NL). McCC represents the mean count of all branching elements within a contract function arising from the implementation of control loop structures and conditional statements (namely if, for, while, do-while loops). NL is a measure of the mean count of the deepest nesting levels of control structures within a contract function. The underpinning idea is that higher are the values of these two metrics, more is the complexity of the smart contract code due to higher average usage of control loops and conditional statements and higher nesting levels of such structures. From Table 5, it is evident that the values obtained as a part of this research are relatively lower than the McCC and NL values obtained from literature. Despite the difference in values being small, it should be appreciated that the authors in their survey already found the distribution of these values to be highly skewed towards lower values for smart contract codes and to be able to attain even lower mean values as compared to those quoted in literature corresponds to a considerable improvement in the reduction of code complexity for smart contracts. This shows that, due to the proper implementation of separation of concerns and object-oriented design patterns in the construction of the CMaaS manufacturing system smart contracts, very low McCC and NL values could be obtained that allowed for the design of smart contract codes with higher gas consumption efficiency.

Table 5
Object-oriented software metrics of Solidity smart contracts for CMaaS manufacturing system.

Metrics	Description	Mean values (this research)	Mean values from literature
LLOC	Logical lines of code	358	36.43
NF	Number of functions	20	4.94
McCC	McCabe's cyclomatic complexity	1.05	1.15
NL	Nesting level	0.12	0.14
NUMPAR	Number of parameters	1.4	1.50
NOA	Number of ancestors	0	1.17
NA	Number of attributes	25	2.47

6. Discussion

In this paper, focus was first placed on the architectural framework of a CMaaS platform that allows mass customization and manufacturing of products by remote clients through cloud centric communications. Work in this paper demonstrated that multi-threaded server-based middleware can be designed and interfaced with existing or legacy software platforms to enable powerful decentralized cloud manufacturing capabilities. It was also demonstrated that cloud-based middleware could very efficiently integrate into and interface with hardware assets or manufacturing nodes. This outcome is important towards the future realization of a completely autonomous manufacturing service that would be intelligent and be composed of a consortium of self-aware machines. Another important potential demonstrated by the work in this research is its contribution in the shifting of the technical and intellectual overhead associated with CMaaS platforms away from the client. Through the implementation of the architectural framework proposed in this paper, remote clients can now practically gain access to the services of a cloud manufacturing entity without having to know much about the intricate details of design and manufacturing, as is often required by end users designing and altering solid models in CAD software often required by conventional CMaaS platforms. Therefore, end-users who are non-experts themselves can participate in design activities – further democratizing access to manufacturing services.

In the second part of this paper, the work proposed an architectural framework and presented design implementation of a decentralized, digital manufacturing system that can work in unison with CMaaS platforms to track product data and manage stakeholders involved in the production, supply and distribution of such parts. Decentralized, digital manufacturing systems have the capacity of providing real time product data for asset tracking. This provides end clients with a history of the product trail within the manufacturing system network of the CMaaS platform enabling provenance verification and product authentication. Interaction between stakeholders across different organizations over the cloud represent ideal cases of distributed systems where different organizations come together and work into the development of a product as is typical on many CMaaS platforms. In addition to providing provenance records through immutable data to clients, blockchain architectures have the capabilities of ensuring the requirements efficiently and securely. Without a central authority controlling the data in a decentralized manufacturing system, participating entities can now interact in a robust environment with high levels of redundancy.

In summary, the contributions that set the work laid out in this paper apart from other research on blockchain based CMaaS platforms covered in section 2 are as follows:

(i) This work introduced the idea of object-oriented design paradigms in the design of complex manufacturing system smart contracts. It was demonstrated that inheritance models of software engineering could be systematically implemented in blockchain smart contract engineering to allow separation of concerns and hence lead to efficient smart contract design under constrained environments.
(ii) It was also demonstrated how different stakeholders in a manufacturing system could be modelled through distinct smart contracts and how inter-contract communication protocols allowed the interplay of such stakeholders within a CMaaS platform. Given the manufacturing system smart contract architecture implemented in this research, CMaaS platforms can now exploit the framework to adapt to scalability issues.
(iii) In order to allow seamless interaction between several stakeholders in the CMaaS blockchain platform and between several software microservices, this paper has shown the design and implementation of plug-and-play middleware software architectures that can easily interface with existing cloud manufacturing infrastructures to allow blockchain based CMaaS capabilities without

introducing significant disruption.

(iv) This paper also demonstrated an appropriate data model for products being tracked by decentralized manufacturing systems. It was showcased through the implementation of the ERC-721 non fungible token standard of the Ethereum ecosystem to model manufacturing product data through the implementation of secure interfaces and by proposing additional methods on the interface to allow transfer and trading of such tokens representing manufactured assets. Having an appropriate data model for digital assets following non-fungible asset protocols will allow the transfer, trading and exchange of such assets to be represented by appropriate events in the decentralized manufacturing system network. This provides a useful guideline to stakeholders into understanding what considerations to make when modelling a digital asset on the blockchain.

(v) This paper also proposed methods and showcased important performance metrics in terms of gas consumption, mining times and cyclomatic complexity of codes that would allow Ethereum based CMaaS platforms to quickly assess the efficiency of their smart contract design patterns prior to deployment. Through appropriate descriptive and inferential statistical methods, it was also validated that the execution of the manufacturing system smart contract methods did not pose any complex requirement in the Ethereum ecosystem as far as transaction fees and mining times are concerned.

(vi) This paper also demonstrated how secure fungible asset transfer models of the global Ethereum blockchain network could be utilized to automate and process secure payments required to settle smart contract conditions as is typical for any manufacturing job on CMaaS platforms.

The work in this paper also addresses several privacy and security issues within the domain of the manufacturing system smart contract. The decentralized Ethereum ecosystem that the smart contracts are based on, already makes sure that a stakeholder's real-world identity is obfuscated via pseudo anonymity [92]. Additionally, the restrictive nature of the Solidity programming language and the presence of internal security features like block gas limits [93] and call-stack depth limits ensure that the transfer of fungible assets across the network happen in a secure environment. The adoption of object-oriented design patterns and separation of concerns principles proposed in this paper enforces an additional layer of abstraction with improved privacy. Such a design paradigm makes sure that the stakeholders can interact with each other through disparate stakeholder and manufacturing system smart contracts. This introduces redundancy and further decentralization since there is now no central point of attack for malicious participants. Should such participants intend to gain access to the information of the stakeholders and breach privacy, they would have to ascertain identity by establishing which stakeholder smart contracts is the manufacturing smart contract communicating with via inter-contract data exchange. These data exchange communications between smart contracts are internal transactions, which are not serialized and are not generated by externally owned accounts. Consequently, this design pattern of encoding manufacturing system and stakeholder smart contracts as disparate entities makes it even more difficult for a malicious participant to breach the privacy of a stakeholder. Additionally, the client interaction middleware designed and showcased in section 4.2 enables the signing of blockchain transactions through secure wallets as opposed to the usage of privacy-compromised, locally stored private keys that can be observed in many contemporary research studies.

This paper also focuses on security, proposing smart contract stop-lossing features of manufacturing system smart contracts in section 4.4 which has the ability of immediately halt transactions across the contracts, should there be any incidence of suspicious fungible asset transfers. Finally, an algorithmic multi-party consensus security protocol for smart contracts was proposed that allows secure smart contract transactions by mimicking multi-sig transaction signature features of the Bitcoin platform. Blockchain based DTs exploit the aspects of consensus algorithms for most of these privacy and security features to work. Consequently, the success of these implementations is reliant on whether there are enough honest participating nodes such that privacy is not compromised. Finally, identities over blockchain based DTs are pseudo anonymous and hence there always remains a chance for malicious participants to link transactions to network addresses and hence breach privacy. While there has been the evolution of emerging security protocols like permissioned controls [94], succinct noninteractive argument of knowledge (zk-SNARK), ring signatures, stealth addresses [95] etc., the choice of the right privacy and security protection protocol for such a manufacturing system is highly dependent on its scale and the intricacies of its functions.

The work in this paper opens several avenues for research in the future. DLTs and blockchains are not the definitive solutions for all the problems that arise from centralized manufacturing systems. The deployment of DLTs have several associated risks that should be taken into consideration. With data distributed among many ledgers and nodes, legal risks associated with DLT deployment remain. For highly regulated manufacturing processes like pharmaceuticals or medical devices, DLT based supply chains can be considered joint ventures where the liability for any transgressions will now be spread across all the participating nodes in the DLT system [96]. Public blockchains like Ethereum provide very low barrier to entry for new participants and since it does not mandate authentication, bad actors can creep in and create challenges for regulation of manufacturing processes. Private or consortium blockchain frameworks can help mitigate those problems, however many of them still suffer from inefficient fungible asset transfer models and do not provide the required commercial confidentiality for business operations [97].

Concerns pertaining to the corruption of data of digital assets in cloud-based systems via infiltrators with malicious intent have been raised in the past and they remain a source of impediment for widespread adoption of cloud enabled technologies in the manufacturing domain. Although blockchain platforms like Ethereum can help with data integrity issues, the sheer magnitude of data generated from manufacturing renders such platforms less effective for high data throughput scenarios. Ethereum based blockchain solutions, due to their restrictive nature and security features are not good data storage solutions. This also limits the capacity of Ethereum based distributed CMaaS platforms to appropriately encapsulate complex product life cycle data as is mandated in modern ISO standards like STEP and STEP-NC. With that in mind, the authors recommend the following future research directions to overcome the existing barriers to widespread adoption of distributed CMaaS platforms among both end users and service providers:

(i) Future work investigating the design interfaces in blockchain based databases like BigChainDB [98] or decentralized file sharing platforms like the Interplanetary File System (IPFS) [99] show promise to tackle the challenges of big-data problems in smart manufacturing. Ongoing research by Bhattacharyya et al. [100] have demonstrated how blockchain based databases like BigChainDB can be utilized to encapsulate more complex data models for manufacturing assets and to automate more complex contractual negotiations involving bidding and real time notifications. This opens avenues for the possibility of inclusion of more complex product data models (like the HOOM [45]) on DLTs that are ISO standards compliant and hence have much wider application prospective. The author envision the design and implementation of hybrid infrastructures and intercommunication protocols and middleware which take advantage of the immutable, big-data capabilities of BigChainDB and the secure, fungible asset transfer models of the Ethereum framework that would lead to the design of more

acceptable CMaaS blockchain solutions.

(ii) Another direction could involve the research of better methods that would make CMaaS platforms more accessible to end users. There needs to be further investigation to find out ways technical and intellectual overheads can be further shifted away from end users. With the emergence of artificial intelligence, machine learning, computer vision and allied technologies, machines and software systems can now be imparted with self-correction algorithms that allow them to take intelligent decisions. The authors envision the application of such modalities into CMaaS client software that would efficiently be able to capture design intent of end users in much shorter times. Pointivo [101] a small business funded by the National Science Foundation has used such technologies to automatically generate 3D model for CAD and building information models.

7. Conclusion

In this work, an architectural framework of an improved CMaaS platform that is capable of doing mass customization of parts from remote clients was proposed and its implementation details was demonstrated. The CMaaS architecture designed was successfully demonstrated to shift technical and knowledge burdens away from remote clients through the implementation of server-based, plug-and-play middlewares that could efficiently interface with existing software and hardware platforms to provide them cloud manufacturing capabilities. Implementation of a decentralized manufacturing system architecture was demonstrated and systematic means of encoding higher efficiency manufacturing system smart contract code constructs through object-oriented and inheritance model paradigms was also described. Additionally, the tracking of digital asset data through decentralized manufacturing systems by modelling them as non-fungible assets was shown and the performance of manufacturing system smart contracts dealing with such assets was demonstrated through statistical inferential tests on test network performance.

Declaration of Competing Interest

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.

Acknowledgements

This work was in part supported by National Science Foundation grant #1764025. Special thanks to Brian Ekins at Autodesk Fusion 360 API team and Nicolas Venturo (OpenZeppelin) for assistance and feedback on the Fusion 360 API middleware and Ethereum 721 non-fungible contract structure respectively.

References

[1] Caggiano A, Segreto T, Teti R. Cloud Manufacturing Framework for Smart Monitoring of Machining. Procedia Cirp 2016;55:248–53. https://doi.org/10.1016/j.procir.2016.08.049.
[2] Tao F, Qi Q, Liu A, Kusiak A. Data-driven smart manufacturing. J Manuf Syst 2018;48:157–69.
[3] Zhang L, Luo Y, Tao F, Li BH, Ren L, Zhang X, et al. Cloud manufacturing: a new manufacturing paradigm. Enterp Inf Syst 2014;8(2):167–87.
[4] Lee J, Bagheri B, Kao H-A. A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. Manuf Lett 2015;3:18–23. https://doi.org/10.1016/j.mfglet.2014.12.001.
[5] Shi J, Wan J, Yan H, Suo H. A survey of cyber-physical systems. International Conference on Wireless Communications and Signal Processing (WCP), November 2011:1–6.
[6] Napoleone A, Macchi M, Pozzetti A. A review on the characteristics of cyber-physical systems for the future smart factories. J Manuf Syst 2020;54:305–35.
[7] Mell PM, Grance T. The NIST definition of cloud computing. 2011. https://doi.org/10.6028/nist.sp.800-145.
[8] Wu D, Greer MJ, Rosen DW, Schaefer D. Cloud manufacturing: strategic vision and state-of-the-art. J Manuf Syst 2013;32(4):564–79.
[9] Ren L, Zhang L, Wang L, Tao F, Chai X. Cloud manufacturing: key characteristics and applications. Int J Comput Integr Manuf 2017;30(6):501–15.
[10] Lu Y, Xu X. Cloud-based manufacturing equipment and big data analytics to enable on-demand manufacturing services. Robot Comput Integr Manuf 2019;57:92–102.
[11] Xu X. From cloud computing to cloud manufacturing. Robot Comput Integr Manuf 2012;28(1):75–86. https://doi.org/10.1016/j.rcim.2011.07.002.
[12] Zhang L, Luo YL, Tao F, Ren L, Guo H. Key technologies for the construction of manufacturing cloud. Comput Integr Manuf Syst 2010;16(11):2510–20.
[13] Coffman, Valerie R., Mark Wicks, and Daniel Wheeler. Methods and apparatus for machine learning predictions and multi-objective optimization of manufacturing processes. U.S. Patent Application No. 10/061,300.
[14] Liu XF, Shahriar MR, Al Sunny SN, Leu MC, Hu L. Cyber-physical manufacturing cloud: architecture, virtualization, communication, and testbed. J Manuf Syst 2017;43:352–64.
[15] Wang L, Törngren M, Onori M. Current status and advancement of cyber-physical systems in manufacturing. J Manuf Syst 2015;37:517–27.
[16] Oliveira T, Thomas M, Espadanal M. Assessing the determinants of cloud computing adoption: an analysis of the manufacturing and services sectors. Inf Manag 2014;51(5):497–510. https://doi.org/10.1016/j.im.2014.03.006.
[17] Rogers EM. Diffusion of innovations. 5th ed. New York: Free Press; 2003.
[18] Tornatzky LG, Fleischer M. The processes of technological innovation. Massachusetts: Lexington Books; 1990.
[19] Riel A, Kreiner C, Macher G, Messnarz R. Integrated design for tackling safety and security challenges of smart products and digital manufacturing. CIRP Ann Manuf Technol 2017;66(1):177–80. https://doi.org/10.1016/j.cirp.2017.04.037.
[20] Tao F, Qi Q. Make more digital twins. Nature 2019;573:490–1.
[21] Iyer K, Dannen C. The ethereum development environment. Building Games with Ethereum Smart Contracts 2018:19–36. https://doi.org/10.1007/978-1-4842-3492-1_2.
[22] Qi Q, Tao F, Hu T, Anwer N, Liu A, Wei Y, et al. Enabling technologies and tools for digital twin. J Manuf Syst 2019.
[23] Wu D, Rosen DW, Wang L, Schaefer D. Cloud-based design and manufacturing: a new paradigm in digital manufacturing and design innovation. Comput Des 2015;59:1–14. https://doi.org/10.1016/j.cad.2014.07.006.
[24] Strickland E. Shapeways bringing 3-D printing to the masses. IEEE Spectr 2013;50(11). https://doi.org/10.1109/mspec.2013.6655831. 22–22.
[25] Wu D, Liu X, Hebert S, Gentzsch W, Terpenny J. Democratizing digital design and manufacturing using high performance cloud computing: performance evaluation and benchmarking. J Manuf Syst 2017;43:316–26.
[26] CloudNC: making machining autonomous. 2020. Retrieved from https://cloudnc.com/.
[27] Plethora. Precision CNC Machining specializing in tight tolerances and complex geometries Retrieved from 2020https://www.plethora.com/.
[28] Fictive - on-demand manufacturing. 2020. Retrieved from https://www.fictiv.com/.
[29] Xometry - CNC Machining, 3D Printing, Sheet Metal & Injection Molding Services. (n.d.). Retrieved from https://www.xometry.com/.
[30] Quickparts. 2014. Retrieved from https://www.3dsystems.com/quickparts/about/quickquote.
[31] Livesource. 2014. Retrieved from https://www.livesource.com/.
[32] Lu Y, Xu X. A semantic web-based framework for service composition in a cloud manufacturing environment. J Manuf Syst 2017;42:69–81.
[33] Wu D, Thames JL, Rosen DW, Schaefer D. Towards a cloud-based design and manufacturing paradigm: looking backward, looking forward. 32nd Computers and Information in Engineering Conference, Parts A and B 2012;Volume 2:315–28.
[34] Esmaeilian B, Behdad S, Wang B. The evolution and future of manufacturing: a review. J Manuf Syst 2016;39:79–100.
[35] Wang Y, Lin Y, Zhong RY, Xu X. IoT-enabled cloud-based additive manufacturing platform to support rapid product development. Int J Prod Res 2019;57(12):3975–91.
[36] Anderson G, Anderson G. The economic impact of technology infrastructure for smart manufacturing. US Department of Commerce. National Institute of Standards and Technology; 2016.
[37] Thompson K. August 23). Digital thread for smart manufacturing Retrieved from 2017https://www.nist.gov/programs-projects/digital-thread-smart-manufacturing.
[38] Helu M, Hedberg Jr T. Enabling smart manufacturing research and development using a product lifecycle test bed. Procedia Manuf 2015;1:86–97.
[39] Fischer K, Rosche P, Trainer A, Feeney AB, Hedberg TD. Investigating the impact of standards-based interoperability for design to manufacturing and quality in the supply chain (No. Grant/Contract Reports (NISTGCR)-15-1009). 2015.
[40] Trainer A, Hedberg T, Barnard Feeney A, Fischer K, Rosche P. June). Gaps analysis of integrating product design, manufacturing, and quality data in the supply chain using model-based defintion. ASME 2016 11th International Manufacturing Science and Engineering Conference. 2016.
[41] Hedberg TD, Krima S, Camelio JA. Embedding x. 509 digital certificates in three-dimensional models for authentication, authorization, and traceability of product data. J Comput Inf Sci Eng 2017;17(1).
[42] Michaloski J, Hedberg T, Huang H, Kramer T, Michaloski J. End-to-end quality information framework (QIF) technology survey. US Department of Commerce, National Institute of Standards and Technology; 2016.
[43] Hedberg T, Lubell J, Fischer L, Maggiano L, Barnard Feeney A. Testing the digital thread in support of model-based manufacturing and inspection. J Comput Inf Sci Eng 2016;16(2).

[44] Helu M, Hedberg Jr T, Feeney AB. Reference architecture to integrate heterogeneous manufacturing systems for the digital thread. Cirp J Manuf Sci Technol 2017;19:191–5.
[45] Bonnard R, Hascoët JY, Mognol P, Zancul E, Alvares AJ. Hierarchical object-oriented model (HOOM) for additive manufacturing digital thread. J Manuf Syst 2019;50:36–52.
[46] Lu Y, Wang H, Xu X. ManuService ontology: a product data model for service-oriented business interactions in a cloud manufacturing environment. J Intell Manuf 2019;30(1):317–34.
[47] Gan Y, He WM, Ihara T. Analysis for the structure of product manufacturing information flow of cloud manufacturing based on information measurement. J Adv Mech Des Syst Manuf 2015;9(3). JAMDSM0043-JAMDSM0043.
[48] Krima S, Hedberg T, Feeney AB. Securing the digital threat for smart manufacturing: a reference model for blockchain-based product data traceability. US Department of Commerce, National Institute of Standards and Technology; 2019.
[49] Lee J, Azamfar M, Singh J. A blockchain enabled Cyber-Physical System architecture for Industry 4.0 manufacturing systems. Manuf Lett 2019;20:34–9. https://doi.org/10.1016/j.mfglet.2019.05.003.
[50] Madhwal Y, Panfilov P. Blockchain and supply Chain management: aircrafts' parts' business case. DAAAM Proceedings of the 28th International DAAAM Symposium 2017;2017:1051–6. https://doi.org/10.2507/28th.daaam.proceedings.146.
[51] Angrish A, Craver B, Hasan M, Starly B. A case study for blockchain in manufacturing: "FabRec": a prototype for peer-to-Peer network of manufacturing nodes. Procedia Manuf 2018;26:1180–92. https://doi.org/10.1016/j.promfg.2018.07.154.
[52] Li Z, Barenji AV, Huang GQ. Toward a blockchain cloud manufacturing system as a peer to peer distributed network platform. Robot Comput Integr Manuf 2018;54:133–44.
[53] Multichain - Enterprise blockchain that actually works. (n.d.). Retrieved from https://www.multichain.com/.
[54] Mandolla C, Petruzzelli AM, Percoco G, Urbinati A. Building a digital twin for additive manufacturing through the exploitation of blockchain: a case analysis of the aircraft industry. Comput Ind 2019;109:134–52.
[55] Huang S, Wang G, Yan Y, Fang X. Blockchain-based data management for digital twin of product. J Manuf Syst 2020;54:361–71.
[56] Vatankhah Barenji A, Li Z, Wang WM, Huang GQ, Guerra-Zubiaga DA. Blockchain-based ubiquitous manufacturing: a secure and reliable cyber-physical system. Int J Prod Res 2019:1–22.
[57] Lemeš S, Lemeš L. June). Blockchain in distributed CAD environments. In international conference "New technologies, development and applications". Cham: Springer; 2019. p. 25–32.
[58] Zohar A. Bitcoin: under the hood. Commun ACM 2015;58(9):104–13.
[59] Dannen C. Introducing ethereum and solidity Vol. 1. Berkeley: Apress.; 2017.
[60] Entriken W. ERC-721 non-fungible token standard. Ethereum Foundation; 2018.
[61] Beal, V. (n.d.). API - application program interface. Retrieved from https://www.webopedia.com/TERM/A/API.html.
[62] Jiao JR, Simpson TW, Siddique Z. Product family design and platform-based product development: a state-of-the-art review. J Intell Manuf 2007;18(1):5–29. https://doi.org/10.1007/s10845-007-0003-2.
[63] Ethereum wallets. (n.d.). Retrieved from https://www.ethereum.org/use/#_3-what-is-a-wallet-and-which-one-should-i-use.
[64] Schaad A, Reski T, Winzenried O. Integration of a secure physical element as a trusted oracle in a hyperledger blockchain. Proceedings of the 16th International Joint Conference on E-Business and Telecommunications 2019. https://doi.org/10.5220/0007957104980503.
[65] Kochovski P, Gec S, Stankovski V, Bajec M, Drobintsev PD. Trust management in a blockchain based fog computing platform with trustless smart oracles. Future Gener Comput Syst 2019;101:747–59.
[66] Sylvester G. E-agriculture in action: blockchain for agriculture: opportunities and challenges. Bangkok: International Telecommunication Union; 2019.
[67] Lomas N. Everledger is using blockchain to combat fraud, starting with diamonds'. Tech Crunch; 2015.
[68] Tian F. A supply chain traceability system for food safety based on HACCP, blockchain & internet of things. 2017 International Conference on Service Systems and Service Management 2017. https://doi.org/10.1109/icsssm.2017.7996119.
[69] Hursch Walter L, Lopes Cristina Videira. Separation of concerns. Technical report by the college of computer science. Northeastern University; 1995.
[70] Rumbaugh JBM. Object - oriented: modeling and design. New Jersey: Prentice Hall; 1998.
[71] Hirai Y. April). Defining the ethereum virtual machine for interactive theorem provers. In International Conference on Financial Cryptography and Data Security. 2017. p. 520–35.
[72] Nielsen JB, Spitters B. Smart contract interactions in coq. arXiv preprint. arXiv 2019;1911:04732.
[73] Grech N, Kong M, Jurisevic A, Brent L, Scholz B, Smaragdakis Y. Madmax: surviving out-of-gas conditions in ethereum smart contracts. Proceedings of the ACM on Programming Languages, 2(OOPSLA) 2018:1–27.
[74] Flask. (n.d.). Retrieved from https://palletsprojects.com/p/flask/.
[75] Rodriguez A. Restful web services: the basics. IBM Dev Works 2008;33:18.
[76] Three.js. (n.d.). Retrieved from https://threejs.org/.
[77] Web3.js - Ethereum JavaScript API. (n.d.). Retrieved from https://web3js.readthedocs.io/en/v1.2.2/.
[78] MetaMask Ethereum Browser Extension. (n.d.). Retrieved from https://metamask.io/.
[79] OpenZeppelin. (n.d.). Retrieved from https://openzeppelin.com/contracts/.
[80] Lin IC, Liao TC. A survey of blockchain security issues and challenges. IJ Network Security 2017;19(5):653–9.
[81] Atzei N, Bartoletti M, Cimoli T. April). A survey of attacks on ethereum smart contracts (sok). International Conference on Principles of Security and Trust. 2017. p. 164–86.
[82] Mehar MI, Shier CL, Giambattista A, Gong E, Fletcher G, Sanayhie R, et al. Understanding a revolutionary and flawed grand experiment in blockchain: the DAO attack. J Cases Inform Technol (JCIT) 2019;21(1):19–32.
[83] Chowdhury N. Consensus mechanisms of blockchain. Inside Blockchain, Bitcoin, and Cryptocurrencies 2019:49–60. https://doi.org/10.1201/9780429325533-3.
[84] Gervais A, Karame GO, Wüst K, Glykantzis V, Ritzdorf H, Capkun S. October). On the security and performance of proof of work blockchains. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016. p. 3–16.
[85] Lamport L, Shostak R, Pease M. The Byzantine generals problem. Concurrency: the works of leslie lamport. 2019. p. 203–26.
[86] Saleh F. Blockchain without waste: proof-of-stake Available at SSRN 3183935 2019.
[87] Parity Documentation - Aura - Authority Round. (n.d.). Retrieved from https://wiki.parity.io/Aura.html.
[88] Abdi H, Williams LJ. Tukey's honestly significant difference (HSD) test. Encyclopedia of Research Design. Thousand Oaks, CA: Sage; 2010. p. 1–5.
[89] Hegedűs P. Towards analyzing the complexity landscape of solidity based ethereum smart contracts. Technologies 2019;7(1):6. https://doi.org/10.3390/technologies7010006.
[90] Tonelli R, Destefanis G, Marchesi M, Ortu M. Smart contracts software metrics: a first study. arXiv preprint. arXiv 2018;1802:01517.
[91] Chidamber SR, Kemerer CF. A metrics suite for object-oriented design. Ieee Trans Softw Eng 1994;20(6):476–93.
[92] Chen J. Hybrid blockchain and pseudonymous authentication for secure and trusted IoT networks. ACM Sigbed Rev 2018;15(5):22–8.
[93] Hartel P, Schumi R. Gas limit aware mutation testing of smart contracts at scale. arXiv preprint. arXiv 2019;1909:12563.
[94] Trump BD, Florin MV, Matthews HS, Sicker D, Linkov I. Governing the use of blockchain and distributed ledger technologies: not one-size-fits-all. IEEE Eng Manag Rev 2018;46(3):56–62.
[95] Henry R, Herzberg A, Kate A. Blockchain access privacy: challenges and directions. IEEE Secur Priv 2018;16(4):38–45.
[96] Zetzsche DA, Buckley RP, Arner DW. The distributed liability of distributed ledgers: legal risks of blockchain. U. Ill. L. Rev. 2018:1361.
[97] Staples M, Chen S, Falamaki S, Ponomarev A, Rimba P, Tran AB, et al. Risks and opportunities for systems using blockchain and smart contracts. Data61. CSIRO), Sydney. 2017.
[98] McConaghy T, Marques R, Müller A, De Jonghe D, McConaghy TT, McMullen G, et al. BigchainDB: a scalable blockchain database (2016) URL 2016https://www.bigchaindb.com/whitepaper/bigchaindb-whitepaper.pdf.
[99] IPFS is the Distributed Web. Retrieved from https://ipfs.io/#how.
[100] Bhattacharyya A, Iyer RS, Ogan KA. SmartChainDB: towards semantic events on blockchains. Academic Presentation 2019. Retrieved from www.cs.toronto.edu/-consens/blocksw/slides/KemaforEtAl.pdf.
[101] Spatial Intelligence for the Physical World. (n.d.). Retrieved from https://pointivo.com/#technology.