IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 15, 2019, accepted December 23, 2019, date of publication January 6, 2020, date of current version January 21, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2964220

Approaching Non-Disruptive Distributed Ledger Technologies via the Exchange Network Architecture

EMANUEL PALM, ULF BODIN, AND OLOV SCHELÉN
EISLAB, Luleå University of Technology, 971 85 Luleå, Sweden

Corresponding author: Emanuel Palm (emanuel.palm@ltu.se)

This work was supported by the *Productive 4.0* Project (EU ARTEMIS JU) under Grant 737459.

ABSTRACT The rise of distributed ledger technologies, such as R3 Corda, Hyperledger Fabric and Ethereum, has lead to a surge of interest in digitalizing different forms of contractual cooperation. By allowing for ledgers of collaboration-critical data to be reliably maintained between stakeholders without intermediaries, these solutions might enable unprecedented degrees of automation across organizational boundaries, which could have major implications for supply chain integration, medical journal sharing and many other use cases. However, these technologies tend to break with prevailing business practices by relying on *code-as-contracts* and *distributed consensus algorithms*, which can impose disruptive requirements on contract language, cooperation governance and interaction privacy. In this paper, we show how our *Exchange Network* architecture could be applied to avoid these disruptors. To be able to reason about the adequacy of our architecture, we present six requirements for effective contractual collaboration, which notably includes *negotiable terms* and *effective adjudication*. After outlining the architecture and our implementation of it, we describe how the latter meets our requirements by facilitating (1) negotiation, (2) user registries, (3) ownership ledgers and (4) definition sharing, as well as by only replicating ledgers between stakeholder pairs. To show how our approach compares to other solutions, we also consider how Corda, Fabric and Ethereum meet our requirements. We conclude that digital negotiation and ownership could replace many proposed uses of code-as-contracts for better compatibility with current contractual practices, as well as noting that distributed consensus algorithms are not mandatory for digital cooperation.

INDEX TERMS Digital negotiation, digital cooperation, digital contracts, smart contracts, distributed ledger technology, blockchain, distributed consensus algorithms, business integration, digitalization.

I. INTRODUCTION

In the wake of Bitcoin's [1] salient rise to prominence [2], other kinds of Distributed Ledger Technologies (DLTs), such as R3 Corda [3], Hyperledger Fabric [4] and Ethereum [5], were created to make the same kind of technology useful not only for cryptographic money, but also for many kinds of *contractual cooperation*. Significantly, Bitcoin enables its users to collectively maintain a tamper-proof ledger, without having to trust in any authority. While the Bitcoin ledger was designed for recording transfers of the Bitcoin currency, ledgers are useful in many contexts where accountability is important, such as when registering ownership transfers, tracking task completions, or managing credits. By being able to reliably maintain ledgers of collaboration-critical data without trusted middle-men, these digital contract solutions enable computers to autonomously initiate and react to cross-organizational interactions, without significant risks for fraud or other types of illicit behavior. In other words, these new DLTs could enable unprecedented degrees of automation across organizational boundaries, with all the economical benefits that would be entailing. This potential has lead to a surge of research interest, with recent works focusing on use cases such as supply chain integration [6], medical journal sharing [7], land ownership registration [8], among many other compelling examples [9].

However, despite the notable appeal of the technology, it has not yet seen any significant degrees of adoption [2], [9]. We believe this is caused primarily by the prevalence of three disruptors, which are (1) contract models based on computer

The associate editor coordinating the review of this manuscript and approving it for publication was Jenny Mahoney.

code, which we refer to as *code-as-contracts*, (2) *distributed consensus algorithms*, as well as (3) latency, throughput and other forms of *performance issues* [10]. In particular,

1) *code-as-contracts* disrupt existing contractual practices by introducing languages that are foreign to the experts now responsible for managing contracts, which makes it difficult and costly to build up organizations that are able to handle them. In addition, no solution we know of allows for the contents of such contracts to be negotiated or renegotiated digitally, which means that they only digitize a smaller part of the collaborative process. Furthermore, they inhibit the application of national laws by making it hard to determine (A) who would be responsible in case of programming mistakes, (B) whether or not a given code-as-contract has been entered into lawfully, (C) the correspondence between computer and legal languages, among other similar issues [11]. Additionally, the use of
2) *distributed consensus algorithms* can disrupt prevalent business practices by requiring groups of parties to validate and vote each others' interactions, even when not directly affected by those interactions. We believe that many organizations will regard this as unacceptable, as it could lead to their competitors retrieving and taking advantage of data about their commitments. Lastly,
3) *performance issues*, often directly related to distributed consensus algorithms [10], inhibit use cases that would require higher interaction throughput or lower latency to be profitable or useful.

In this paper, we show how these disruptors can be avoided via the application of our *Exchange Network* (EN) architecture. In particular, the architecture describes how to design cross-organizational systems that provide *negotiation and ownership* as primary abstractions rather than function invocations and state machine transitions. Consequences of our design are (1) supporting code-as-contracts becomes optional, (2) the negotiation of contract creation, amendment and exceptions can be digitized, as well as (3) the concrete type of used consensus model becomes non-essential, which means that such can be chosen that is more compatible with current business practices. To establish the adequacy of our architecture, we first formulate six requirements for effective contractual cooperation, after which we describe how our own implementation of the architecture fulfills those requirements to a degree comparable to other DLTs.

The rest of the paper is organized as follows. In Section II, we relay why this paper was written and mention prior art. In Section III, we outline how existing DLTs (1) manage without code-as-contracts, (2) seek compatibility with legal institutions and (3) avoid disruptive consensus procedures. In Section IV, we characterize contractual cooperation and derive six requirements from that characterization. In Section V, we describe the EN architecture, our implementation of it and an illustrative use case. In Section VI, we consider how our EN implementation, R3 Corda [3], Hyperledger Fabric [4] and Ethereum [5] fulfill the requirements we present in Section IV. Finally, in Sections VII and VIII, we discuss the implications of our work and relay our conclusions.

II. BACKGROUND AND PRIOR ART

This paper is written as part of the Productive 4.0 project,[1] which is a multidisciplinary research effort aimed at making industries more digital and interconnected. Important goals of the project are (1) improved supply chain integration, (2) enhanced product life-cycle management and (3) digitalized production.[2] The fulfillment of all three of these goals will eventually require that contractual and financial assets can be digitally represented and managed by, as well as transferred between, the stakeholders of industrial value chains.

We have worked together with other project participants, representing Volvo Group, SEB, NXP and other companies,[3] to better understand the real-world circumstances in which these cross-organizational systems would have to exist. One of the fruits of this collaboration is the EN architecture, which we presented previously in [12]. That first paper focuses on the architecture itself, the implications of implementing it with different consensus models, and how it could be used by Volvo Trucks. However, it does not thoroughly present one of our most significant findings, which is that the prevailing code-as-contracts and consensus model, first popularized by Ethereum [5], is not nearly as essential to digital contractual collaboration as we perceive many to believe. To make this case as rigorous as possible, we here reiterate some EN concepts, give more details about our EN implementation, as well as presenting an extensive qualitative analysis.

III. RELATED WORK

However, not all DLTs have the contractual, consensus or performance issues we mention in Section I. Here, we consider how others have avoided or addressed those issues.

A. DLTs WITHOUT CODE-AS-CONTRACTS

Bitcoin [1] provides a stack-based language called *Script*,[4] limited to specifying programmatic spending conditions. Cryptonite [14], originally created from the Bitcoin source code, omits *Script* support completely to simplify the pruning of older transactions from the blockchain it maintains. Both of these systems facilitate counter-parts to electronic debit cards, making their purpose and usage familiar to many, even if they are slower and less deterministic [10].

However, the reason these systems are able to function without code-as-contracts, or any other equivalent, is because

[1] The project website is currently available at https://productive40.eu.

[2] A more exhaustive description of these three goals, or pillars, can be read at https://productive40.eu/2019/07/10/three-pillars-parts-of-one-solid-wall.

[3] A list of all partners is available at https://productive40.eu/stakeholders, which includes the companies we refer to. We name key contributors to this work in the Acknowledgments section, located towards the end of the paper.

[4] As *Script* is not specified in the Bitcoin paper, interested readers might want to consult [13], which contains a formal description of the language. Note that we consider *Script* too limited to be a code-as-contracts system.

they are application-specific and, consequently, do not need to support more complex collaborations. The architecture and implementation we present facilitate contractual cooperation via negotiation and ownership exchanges, which we argue could be practically equivalent to conventional contracts.

B. DLTs SEEKING LEGAL COMPATIBILITY

The Ethereum blockchain system claims to support *Smart Contracts* [5], which were originally described by N. Szabo in [15]. In the case of Ethereum, such a contract is a set of computer instructions that are executed as decided by the majority vote of the Ethereum network. A significant limitation of such a system, however, is that the Ethereum majority wields no other power than deciding what to append to the immutable ledger it maintains. If anything bound to an external domain, such as the physical world, is out of order, the majority is unable to mitigate it without assistance. For example, the majority is not able to detect whether sold goods are being physically transported to their buyers. While national legal institutions are available for trying and correcting contractual deviations in other human domains, it is currently unclear how and if such institutions will consider smart contracts and their ledgers as evidence [11].

To improve compatibility with existing legal instances, some distributed ledger systems allow for the association of legal prose with their code-as-contracts. For example, R3 Corda [3] allows its *state objects*, which encode the intents of Corda's transactions, to refer to both legal prose *and* contract code. Such state objects are exchanged in accordance with predefined patterns, referred to as *flows*, which are defined in a programming language. Another related example is the *Ergo* programming language of the *Accord project* [16]. The language facilitates the creation of contracts that are both code-based and legal that can be executed by smart contract systems, such as Hyperledger Fabric [4].

However, referencing or integrating legal prose is not the same as making code-as-contracts optional. Both of these systems rely on domain models in which the state of a distributed computer is updated through the execution of functions. Consequently, the programming of states and functions becomes critical to the formulation of their contracts. Further, ensuring a contract is legally compatible should require that each computer state can be mapped to legal rights and obligations in such a way that any relevant third-party institution can know the standing of each party.

In contrast, the architecture we propose in Section V-A relies on a domain model of negotiated ownership exchanges. Consequently, defining the implications of token ownership becomes the primary concern of a contract maker, not the programming of states or functions. Each token ownership could be seen as symbolizing a set of rights and obligations, which means that determining the legal standing of each relevant party becomes an exercise of determining the ownership history of each relevant token. While our domain model perhaps could be extended by superimposing state machines or any other code-as-contract capabilities, it is significant that our system *can* be used without such capabilities. We assume that a simpler system stands a better chance of being tried successfully in national courts of law and receive business adoption, for which reason it we deem it relevant to focus primarily on essential functionality, however useful other features may be.

C. DLTs AVOIDING DISRUPTIVE CONSENSUS MODELS

While the distributed consensus algorithms often employed by blockchain systems and other DLTs may make them practically resilient to certain kinds of attacks, it also makes them expose sensitive information to network participants not directly concerned [17]. Additionally, distributed consensus can be a time or resource intensive activity, significantly impacting interaction throughput and latency [10].

Instead of trying to mitigate these issues directly, R3 Corda avoids them by not requiring a distributed consensus algorithm to be used by normal nodes [3]. Consensus is reached either between pairs of peer nodes, which does not necessitate the use of complex algorithms, or within pools of *notary nodes*, which do have to use distributed consensus algorithms. Notary pools are tasked, when requested by normal nodes, to ensure *state objects* cannot be consumed twice, which ensures that transferable assets cannot be duplicated by sending them to more than one recipient, among other things. As the notaries only see the cryptographic hashes [18] of the state objects they are given, they are unable to inspect the data of the state objects they validate. Note that when not dealing with assets that can be transferred multiple times, notary nodes do not have to be used at all.

The system design we propose in Section V relies on the same fundamental approach to consensus as R3 Corda, by which we imply that interactions do not need to be relayed through a network of reviewing voters and that ledgers are replicated primarily between pairs of nodes.[5]

IV. CONTRACTUAL COOPERATION

How can one tell whether or not a given system adequately and effectively facilitates contractual cooperation? To be able to answer this question to a satisfying degree, we here present a characterization of such contractual cooperation, as well as a list of six requirement, formulated with that characterization in mind.

A. CHARACTERIZATION

a) **What is contractual cooperation?** It is the coming together of free agents into a *joint undertaking*, in which each agent is incentivized to collaborate by it somehow contributing to the fulfillment of the agent's *own* goals or ambitions. In other words, contractual cooperation takes

[5] Due to the similarities between our implementation and R3 Corda, one might wonder why we did not create it by writing a so-called *CorDapp* [3], or by modifying the R3 Corda code base. The answer is that the code-as-contract system of R3 Corda makes up a major part of its code base, and we believe that replacing it or creating a layer on top of it would have introduced more complexity than our from-scratch implementation required.

place in a setting where different parties take advantage of each other's capabilities for the sake of promoting the realization of their own distinct ends, which may or may not be conflicting. The undertaking is formalized by having each involved agent accept a contract, which states what *rights and obligations* each participant will have and how these can change.

b) **What problems characterize such cooperation?** More than any other, we believe it to be *uncertainty about the incentives of the counter-parties*. Cooperation takes place in a volatile world where circumstances change, suddenly or gradually. New competitors emerge, laws change, and trends shift, which could make prices drop, new markets become available, or existing businesses unprofitable. Such events might make any counter-parties want to discontinue their involvement or change their terms. Other potential problems may include counter-parties being or becoming fraudulent, incapable of fulfilling their roles, unaware of key limitations, or leaking sensitive facts to competitors.

c) **How are those problems mitigated?** Through the use of different kinds of *risk aversion strategies*. A common such is ensuring misbehaving parties can be *compelled* to conform or compensate its counter-parties, which can be accomplished by agreeing on an adjudicator when a contract is first accepted. Examples of such adjudicators could include courts of law, private arbitration firms or various kinds of member councils. A related strategy is to *prevent* or *disincentivize* misbehavior by letting trusted third parties act as mediators, controlling sensitive exchanges or other interactions. When a DLT relies on a distributed consensus algorithm, which we take a critical stance towards in this paper, its user base collectively forms a mediator that is guaranteed to act only on the majority's behalf. However, for such a mediator to be effective, the majority must have enough power to prevent all relevant kinds of misbehavior, as we note may be problematic in Section III-B. Other possible strategies could involve the continual assessment of trustworthiness, the use of insurance agreements, or the concealment of significant facts from counter-parties and competitors, which otherwise would be able to use that information to the detriment of the concealing party.

B. REQUIREMENTS

Having established what contractual cooperation *is*, we now proceed to formulate requirements we know must be satisfied in order for a given *system* to support it effectively.[6]

1) *Negotiable terms.* Each agent joins a given undertaking because it believes it to help it achieving its own ends. To reach the point where all candidate agents consider a potential cooperation as advantageous, there will likely have to be room for its terms to be negotiated. Furthermore, as circumstances can change after a collaboration has been formalized, there will likely have to be room for those terms to be renegotiated later. In other words, a system for effective contractual cooperation should be able to support the negotiation of (1) contract creation, (2) contract amendments and (3) contractual exceptions.
2) *Consistent interpretation.* There must be some kind of framework in place that guarantees that each contractual partner interprets the terms of the contract the same way, especially considering what right and obligations are associated with each party at every given instance. Even if adjudicators can be asked to judge in case of interpretations differing, as we consider further down, it does not remove the need for interpretation to be sufficiently consistent for collaboration to be possible to begin with. In other words, a sufficiently well-defined legal vocabulary must be shared before any contract can be meaningfully commited to.
3) *Interactional privacy.* Knowledge about the activities of one's competitors is a significant means to improve the effectiveness of one's own business strategies. This means that ensuring contracts and cooperations remain concealed from competitors can be key to preventing that information from affecting their competitiveness. It could, for example, involve the obligation of each party to not reveal certain facts about a contract, or the use of cryptographic means to ensure messages remain obscure to potential observers.
4) *Provable acceptances.* Being able to prove that an agreement has taken place gives each party some power over its counter-parties, in the case anyone would break the terms of that agreement. Those proofs could, for example, be used in a court of law, or be shown to others to deter them from collaborating with the offender. Note that with the term *acceptance*, we refer not only to the signing of a contract, but to any interaction where the contractual standings of two or more parties change. Such changes in standing may occur when contractual clauses are fulfilled, such as by delivering a good.
5) *Effective adjudication.* Having access to adjudication is paramount both because it may allow disputes to be resolved and because it could deter any cooperating parties from deviating from their obligations. However, it requires that the adjudicator, whether it be a court of law or anything else, is able to (1) access, interpret and consider relevant evidence, as well as (2) compel the party deemed at fault to make reparations. This may require that a given agreement is considered lawful by the adjudicator, which could, for example, include proof that a voluntary offer and accept has taken place [11]. If the adjudicator is a computer system, such as a network of Ethereum [5] nodes, it must be able to access whatever power is required to enforce its judgements.

[6] The list we present reflect what we considered most relevant to cover in this paper. It is by no way complete. For example, we do not consider the economic incentives of participation, message integrity guarantees, or having reasonably synchronized clocks.

6) *Trustworthy identification.* Contractual partners have to be able to reliably determine if any received request originated with their counter-parties or not, as a means of avoiding fraud by malicious third parties. Additionally, adjudicators need to be able to assert that signatures, or other attestations, are authentic and refer to legally relevant entities. In a technical setting, this may involve being able to map legal entities to certain kinds of cryptographic primitives, such as public keys.

V. THE EXCHANGE NETWORK ARCHITECTURE

Having established what contractual cooperation is, to a workable extent, as well as some key requirements for such a cooperation to be effectively executed, we are now ready to describe how we believe the collaborative process could be made digital. Concretely, we outline our implementation-independent EN architecture,[7] present how we went about to implement it, and then describe a simplified use case, by which we demonstrate the utility of our architecture.

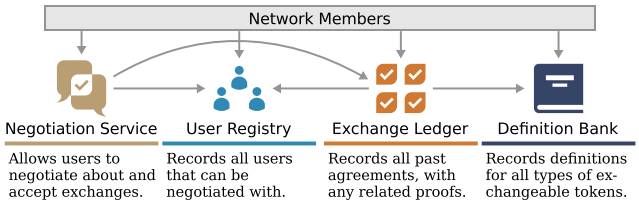

FIGURE 1. EN components. Arrows point from the using to the used component, which, for example, means that the NS makes use of both the UR and EL components. Each component also provides some kind of interface for eligible network members.

A. ARCHITECTURE OVERVIEW

An EN[8] can be a monolithic or a distributed application, facilitating a digital marketplace where well-known types of assets can be negotiated about, exchanged, and proven to have been part of past exchanges. More specifically, an EN facilitates coordinated changes to the owners of digital *tokens*, each of which could be thought of as symbolizing certain rights or obligations, such as the right of ownership, the obligation to render a service to a certain other party, or the right to receive payment for the fulfillment of a specific task. The architecture consists of four primary components, illustrated in Fig. 1, which are

1) the *Negotiation Service* (NS),
2) the *User Registry* (UR),
3) the *Exchange Ledger* (EL) and
4) the *Definition Bank* (DB).

Before presenting each of those components in turn, we want to stress that we make no assumptions about how they store data or coordinate user interactions, as long as data can be accessed and members interact. The components fulfill abstract functions that can be realized in multiple ways. However, we only describe one way of implementing the architecture in this paper, which is intended to demonstrate a non-disruptive approach to designing systems for digital collaboration. In [12], we also consider how a blockchain system like Hyperledger Fabric [4], or a common database system like MySQL [19], could be used as implementation foundations.

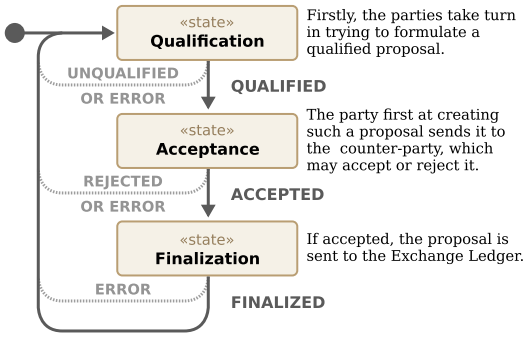

FIGURE 2. A naive state machine, illustrating how two negotiating parties could progress from an initial proposal to an accepted and finalized such.

1) NEGOTIATION SERVICE

The NS allows the members of an EN to propose, accept and reject exchanges of tokens, and submits any mutually accepted exchanges to the EL component. While this could be facilitated via FIPA00037 [20] or some other alternative, we present our own protocol here. The benefits to having our own protocol are largely educational, as it allows us to present something less complex that can be easily understood by researchers with many kinds of backgrounds. Concretely, our protocol lets EN users negotiate by sending *proposals* to each other. Each negotiation progresses through three phases, (1) *qualification*, (2) *acceptance* and (3) *finalization*, as depicted in Fig. 2 and described below.

1) *Qualification.* The first objective of a negotiation is to find a *qualified* proposal believed to be acceptable to each party. A qualified proposal is such that leaves no room for ambiguity regarding who would own what tokens if the proposal would be accepted (i.e. it is a valid *offer*). The proposal is searched for by having the negotiating members take turn in trying to formulate it. If not enough information is had for a candidate proposal to be qualified, an unqualified such may be used instead. Unqualified proposals may refer to abstract types of tokens, include choices, or identify undesired tokens. To facilitate the communication required to send these proposals, the *Proposal* message in Fig. 3 is provided.
2) *Acceptance.* As soon as one party formulates a qualified proposal, the objective becomes to determine if the counter-party deems it acceptable. After having sent the qualified proposal, the counter-party either rejects

[7] See [12] for a more complete description of the EN architecture.

[8] Inspired by the term *social network*, we chose the name *Exchange Network* for our architecture. The name is intended to invoke the idea of an ever-changing network of interacting actors, primarily concerned with the negotiation and exchange of goods, services, or other values.

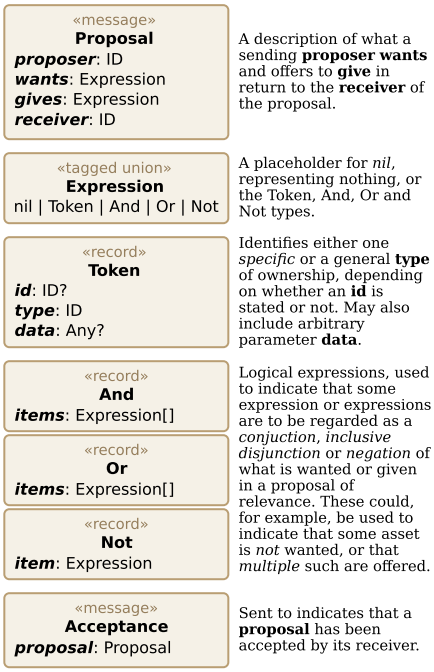

FIGURE 3. **The Proposal and Acceptance messages, with associated data types. ID represents an arbitrary identifier type, question marks (?) are used for optional values, while brackets ([]) are used to denote array types.**

it by sending a new counter-proposal, or accepts it using the *Acceptance* message in Fig. 3. If rejected, the negotiation returns to the **Qualification** phase.

3) *Finalization.* When a qualified proposal has been both formulated and accepted, it is submitted by the NS to the EL. The parties are notified when it is known whether that submission succeeded or failed, after which the negotiation returns to the **Qualification** phase. If there is more to negotiate about, negotiation continues. In any other case, the parties are free to terminate the negotiation session.

2) USER REGISTRY

The *User Registry* (UR) is responsible for associating the *internal* identity of each EN member with its *external* identities. An internal identity is an identifier used to refer to an EN member within the system, such as in proposals, acceptances or exchanges. External identities, on the other hand, is what allows for members to recognize other members outside the bounds of the EN. In whatever manner a given UR component is implemented, be it a database of x.509 certificates integrating with some public-key infrastructure [21] or something completely different, it must be able to guarantee that the identities of all members are trustworthy.

3) EXCHANGE LEDGER

The EL conceptually maintains an append-only ledger of **Exchange** records, each of which consist of an *Acceptance*, as depicted in Fig. 3, and any other data of relevance. As a consequence, the EL can be used by EN members to (1) determine if proposed or already finalized ownership exchanges are *sound*, and (2) prove that past ownership exchanges have taken place. Soundness can be determined by ensuring the tokens of a proposal adhere to their *tests*, which may include taking historic exchanges of relevance into account. Soundness is described further in Section V-A.4. The EN architecture makes no assumptions about how past ownership exchanges are proven to have taken place, as long as they can be. However, we consider one concrete way such proofs can be facilitated when we consider our implementation in Section V-B.

4) DEFINITION BANK

The last component, the DB, defines the implications of owning or creating every known type of token, especially in terms of what may be done with them and any entities they represent. A DB could be regarded as a dictionary, allowing EN members to look up **Definitions**, as defined in Fig. 4, by their names, hashes, or other identifiers.

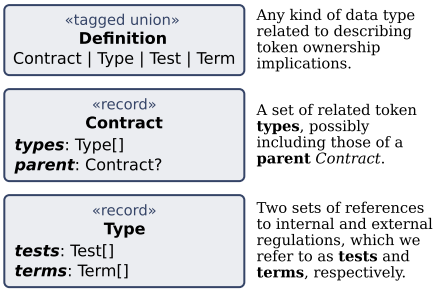

FIGURE 4. **Some proposed kinds of DB definitions. Our naive Contract contains only Types, implying it could be useful for direct machine-verification of contractual events if any of those Types would refer to Tests.**

Concretely, the purpose of maintaining definitions is to ensure *soundness* can be verified for proposed and finalized exchanges. Soundness is established by asserting that a given exchange adheres to both internal and external *regulations*.

- *Internal Regulation.* These regulations, which we also refer to as *tests*, ensure tokens cannot be abused inside an EN. A test could be thought of as a function taking a proposal, an EL and a DB as arguments, returning *true* only if the proposal is sound. For example, tests could limit the number of times a certain type of token can change owner, restrict creation or ownership of specific tokens to a fixed set of eligible members, or set expiration dates after which some tokens may no longer be exchanged. In other words, they could be used to prevent some unsound ownership exchanges from taking place at all.
- *External Regulation.* These exist to ensure any entities represented by any EN tokens are not abused outside the bounds of the EN. We refer to these regulations as *contractual terms*, or just *terms*, and they may or may not be machine-readable. For example, let us assume two EN members have exchanged one token representing the right to a vehicle repair for another representing a promise of payment. At this point, there is no way for

the EN itself to determine if any vehicle is repaired or any payment is made, as these events happen outside the computers of the EN. This could be mitigated by ensuring the types referenced by the exchanged tokens contain contractual terms, in the form of legal prose, honored by some legal authority. As long as the finalized exchange itself counts as proof, an appeal could be made to that authority to resolve any disputes.

B. IMPLEMENTATION

While possible to implement an EN in many different ways, we were particularly interested in doing so such that the existing contractual paradigm could be preserved, as far as reasonably possible. For this reason our implementation[9] does not use any kind of distributed consensus algorithm, and neither does it support any kinds of code-as-contracts, even if we define machine-executable verification functions, or *tests*, as part of the architecture in Section V-A.4. In fact, our implementation is made from scratch. It is not built on R3 Corda [3] or any other existing DLT solution. Concretely, the implementation is intended to mimic the way common paper contracts and other forms of signed instruments are used, which are known only to two or more parties until the event of a dispute, in which case those instruments are revealed to a court of law or other adjudicator.

We begin the description of our implementation by giving an overview of its design, including its user and application interfaces, after which we describe the data structure it uses to construct non-repudiable ledgers, and, finally, present a limited scenario that our implementation can run.

1) DESIGN OVERVIEW

Our system consists of a *node* both serving a web client and communicating with other nodes, as depicted in Fig. 5.

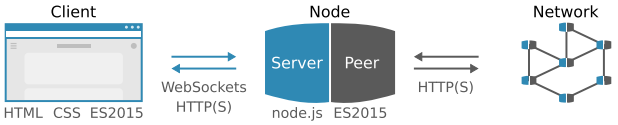

FIGURE 5. The general design of our EN implementation. Humans operate each *Node* using a web browser *Client*. Every *Node* uses an internal *Server* to both provide its *Client* with static HTTP(S) [22] resources and send runtime data via WebSockets [23]. Each *Node* also contains a *Peer* module, which is used to communicate with the *Peer* modules of other *Nodes* over HTTP(S). The design requires no central data repository or any centralization of control.

The business logic of both the node and the client it serves are programmed in the TypeScript programming language [24], which compiles to ECMAScript 2015 [25], also referred to as JavaScript (JS), before execution. The JS of the node is executed by the node.js runtime [26]. The visual structure, styling of the client application are defined using HTML [27] and CSS [28], respectively, and must be executed via a web browser complying to the referenced standards.

[9] Source code, as well as installation and evaluation instructions, are available at https://github.com/emanuelpalm/en-signature-chains-poc. The paper describes commit `694e3a73a1fbae67b9c106d47bd5a1`.

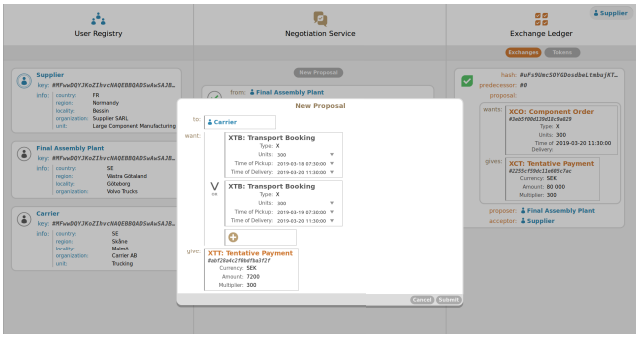

FIGURE 6. The three columns of the client user interface, behind a dialog in which a new proposal is formulated. The formulated proposal is a request to a carrier to transport 300 components for 7200 SEK per component, while giving the carrier the option of choosing between two pick-up dates. The screenshot is taken from the demo application in the implementation code repository. See Footnote 9 on page 12385 for details on how to access the code and instructions for running the demo. The scenario it illustrates is described in Section V-B.3.

The client, shown in Fig. 6, divides it user interface into three columns, (1) User Registry, (2) Negotiation Service and (3) Exchange Ledger, which correspond to three of the EN components. The first column lists trusted user identities, the second column allows sending and accepting proposals to and from other users, while the third column lists all known finalized ownership exchanges. A template token system is provided as a form of naive DB component, which ensures that created tokens follow configurable rules.

While human users communicate with their nodes using web clients, the nodes themselves communicate with each other by sending HTTP(S) requests to the endpoints outlined in Table 1, which each node is expected to expose.

TABLE 1. HTTP(S) endpoints exposed by nodes via their *Peer* interfaces.

Method	*Path*	*Description*
POST	**/proposals**	Submit a signed exchange **Proposal**.
POST	**/acceptances**	Submit a signed and *qualified* proposal **Acceptance**.
POST	**/exchanges**	Share a known previously finalized **Exchange**.

In particular, the node is designed such that if a sent proposal includes a token containing a reference to a previous exchange, it will automatically send that exchange before the proposal. This means that finalized exchanges can be distributed to third parties as proof that something has been accepted by a third party. This is utilized in the example use case in Section V-B.3 by a carrier, allowing it to prove to its client that a delivery was accepted by its recipient.

Our implementation exists to fulfill three functions, (1) to force us to confront and reevaluate the EN and SC concepts during its development, the result of which is this paper, (2) to confirm that our concepts are complete enough to be implemented, as well as (3) to give us a tangible artifact to demonstrate to industry experts in order to receive relevant feedback. None of these three functions require having a production ready system, for which reason we made some important delimitations to reduce implementation effort. For example, any UR and DB data must be provided at node startup, and cannot be changed or added to during runtime.

Even though messages between peers are signed and verified cryptographically against known users, no communication transports are encrypted. Client users are not authenticated or authorized by their nodes. Finally, while HTTP endpoints are provided for sending proposals, acceptances and exchanges, there are no such for requesting exchanges, definitions, users or other relevant data.

2) SIGNATURE CHAINS

To ensure that finalized exchanges can be proven to have taken place, our implementation relies on data structures that use signatures and hashes in a way comparable to blockchains. However, as they do not gather records in batches, we instead refer to them by the name *Signature Chains* (SCs). Concretely, an SC consists of a chain of records, each of which *may* refer to (1) a previous record and (2) a definition of relevance. Each record is cryptographically signed by one or more *attestors*, in our case a proposer and acceptor, and any references to records or definitions are the cryptographic hashes of the data referred to [18]. By implication, a third party given a chain of records, with any associated definitions, becomes able to verify that the records

1) indeed have been signed by their attestors,
2) were created in a certain order, and
3) always have referred to the provided definitions.

Rather than SCs being stored in a centralized or replicated repository, each possible pair of EN members may maintain their own sets of chains, as depicted in Fig. 7. This leaves room for every such pair of members to maintain privacy, given that they can agree on not sharing their mutual records with others. By implication, it also means that both members of each pair can independently reveal any shared chain to any party of interest, such as a court of law, partner, or other party.

In Signature Chain ENs, each possible member pair may maintain its own set of SCs. Each such potential set is here denoted by a line connecting two members.

FIGURE 7. The potential sets of SCs, or ledgers, in a six user EN.

To concretely implement the SC data structure, we amend the **Proposal** and **Acceptance** types as described in Fig. 8.

As our particular implementation does not provide a DB component, but a simpler template system assuming tokens are defined elsewhere, the **definition** field of all **Proposals** is always empty. This limitation is not critical for our purposes, as we can assume that the set of used templates, and their associated legal interpretations, are known beforehand by all participants.

In the case of a fully implemented DB component being available, however, then all definitions ought to refer to their subdefinitions via their hashes. This would guarantee that those definitions cannot be modified without it being detectable. An example of a SC with definition references is illustrated in Fig. 9.

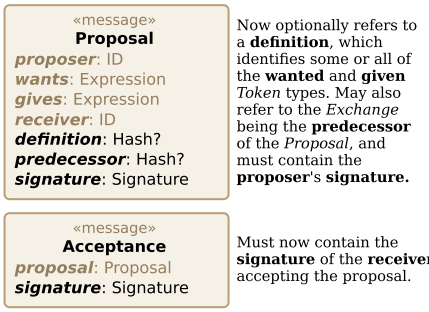

FIGURE 8. Amended variants of messages first outlined in Fig. 3.

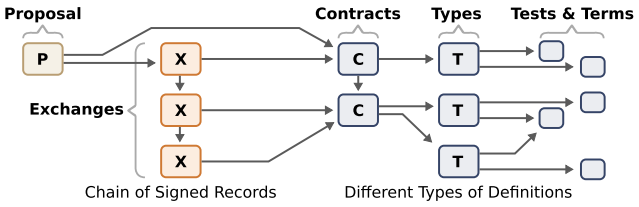

FIGURE 9. An example SC. Arrows denote references by hash. Assuming that proposals and exchanges are signed, any party trusting the identity of the signatures can verify that any associated definitions are not modified since the proposal or exchange was signed. This requires, however, that the verifying party can access referenced exchanges and definitions.

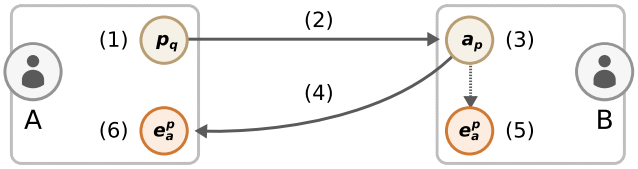

FIGURE 10. The six steps, taken by *A* and *B*, from the creation of a qualified proposal to its acceptance and finalization. Circles denote artifact creation, solid arrows artifact transmission and the dashed arrow an internal transition. Both *A* and *B* must cryptographically verify the artifacts they receive.

SCs are created or extended through a procedure of six steps, beginning at the point where a **Proposal** is formulated by some party A that will subsequently be accepted by its receiver B. We describe the steps in Fig. 10 and below.

1) A creates and signs the qualified **Proposal** (p_q).
2) A sends p_q to B.
3) B creates and signs the **Acceptance** (a_p) from p_q.
4) B sends a_p to A.
5) B puts a_p and its hash into an **Exchange** (e_a^p).
6) A puts a_p and its hash into an **Exchange** (e_a^p).

While a benefit to this procedure may be its lack of complexity, it does have the following potential weaknesses.

- *Acceptance asymmetry.* A is unaware that B accepted and signed p_q unless step (4) is completed successfully. If B is malicious, it could perhaps be advantageous for B to create a_p while concealing it from A.
- *Clock divergence.* In certain scenarios, a p_q may have to be accepted within a certain time window. As each party has its own clock, there could be diverging opinions on whether or not an acceptance (a_p) is timely.

- *Exchange malleability.* While p_q and a_p are signed, e_a^p is not. This gives room for both A and B to record whatever private data they want to associate with a_p in e_a^p, but it also means that if e_a^p is distributed, any fields other than a_p are malleable without it being noticeable.
- *Token duplicability.* In cases where tokens needs to be transferable multiple times, it becomes possible for a malicious party to transfer the same token to multiple counter-parties without it being immediately detectable. As information only is shared as strictly needed, it will seem to both as if they now become the legitimate owners of the new token, which, for example, could be tied to the ownership of a physical good.

All of these weaknesses can be countered, however, through different uses of trusted third parties. And if trusting a single third party would be an issue, for reasons such as concerns about trustworthiness, privacy or fault-tolerance, voting networks of third parties could be a viable alternative.[10] For a concrete example of how a single trusted third party could be used, consider the procedure in Fig. 11.

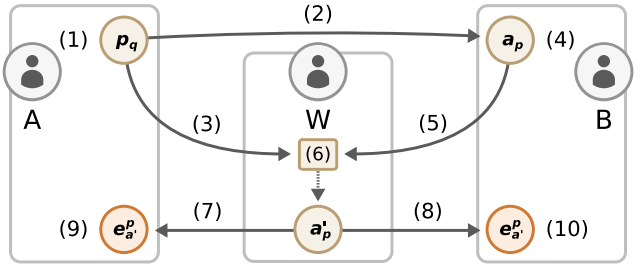

FIGURE 11. A variant of the steps described in Fig. 10. Here, a trusted third party W is provided with the qualified proposal (p_q) in step (3) and the acceptance (a_p) in step (5). If a_p contains p_q and seems to be valid, W creates and signs a'_p in step (6) and then sends it to both A and B.

A trusted third party (W) could assert the timeliness of the acceptance (a_p), ensure both A and B receive the signed acceptance (a'_p), as well as remember whether or not a token part of a proposal has been transferred before.

One or more parties being witnesses of an exchange could also have desirable contractual implications. For example, an insurance agency being a witness, and thereby be given the opportunity to ratify an exchange, could be used as a way of ensuring the insurance agency remains commited to some prior insurance agreement. Another example could be a situation in which some token is owned by multiple parties. All parties except for the one initiating the transfer of the token could be called upon as witnesses to ensure they all ratify it being exchanged.

Before we continue, we want to note that the reason for our showing the extended exchange protocol is to establish that the EN architecture *is able* to facilitate the use of witnesses, which we deem to be a critical requirement for many relevant use cases. We do not claim that witnesses is a unique or somehow special feature of either our architecture or our implementation of it. Rather, many existing smart contract systems, such as Ethereum [5], Hyperledger Fabric [4], and R3 Corda [3], do support these kinds of setups by defining them in their contract programming languages.

3) EXAMPLE USE CASE

As our implementation was designed to demonstrate the EN and SC concepts, it comes with a set of files for running an example use case.[11] We here proceed to describe that example scenario, as it gives another perspective on how our implementation is designed to work. The example consists of six interactions between three partners, as described below and in Fig. 12, using the tokens in Fig. 13.

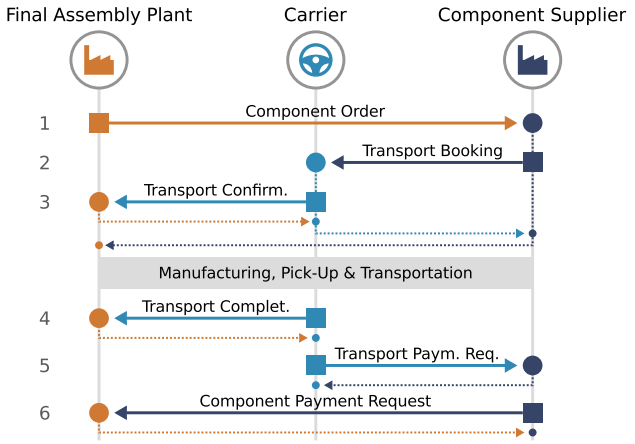

FIGURE 12. The six steps of the example use case. Solid arrows represent sent proposals, while dotted arrows represent sent proposal acceptances.

The goal of the following interactions is to have certain components manufactured and delivered from a *Supplier* (S), via a *Carrier* (C), to a *Final Assembly Plant* (A).

1) *Component order.* A sends a proposal to S, wanting a component order of 200 units, which are to be delivered at a certain date. In return, A offers a tentative payment of 100 000 SEK per component.
2) *Transport booking.* S sends a proposal to C, wanting a transport booking for the components and the delivery time requested in (1). In return, S offers a tentative payment of 7 800 SEK per transported component.
3) *Transport confirmation.* C sends a proposal to A, in which C requests that A confirms the transportation in (2). When accepted by A, C proceeds to also accept the proposal in (2), and S then accepts the proposal in (1).
4) *Transport Completion.* C sends another proposal to A, wanting A to confirm that the transportation accepted in (3) has been completed, which is then accepted by A.
5) *Transport Payment Request.* C then sends the exchange finalized in (4) together with a proposal of payment

[10] This is effectively what R3 Corda achieves via their *notary pools* [3] while preserving a significant level of privacy, as described in Section III-C.

[11] See Footnote 9 on page 12385 for a link to the source code repository, which also contains detailed instructions for running the demo.

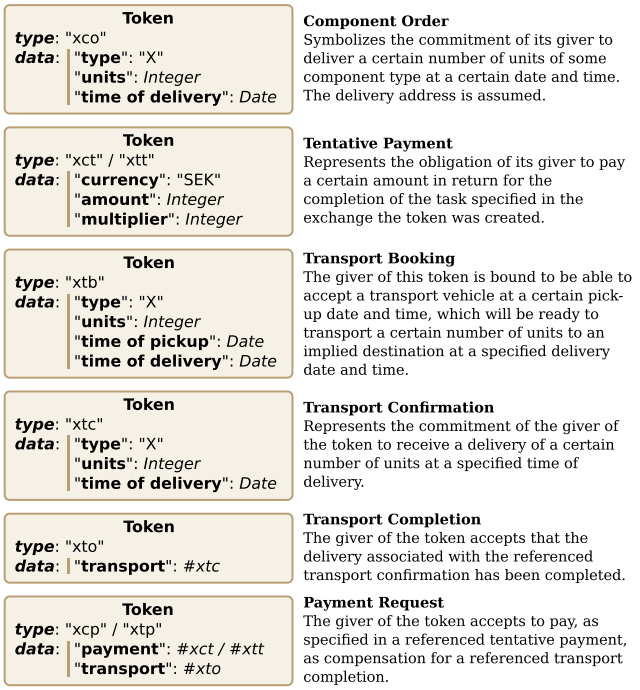

FIGURE 13. Informal definitions of the token types used to facilitate the example use case, with technical descriptions on the left and legal implications on the right. Two of the tokens exist in two type variants each, useful only to allow our implementation determine how to automatically populate certain data fields. Compare with the Token type definition in Fig. 3.

to C, in which C refers to the transport completion in (4) and the tentative payment in (2). S accepts.

6) *Component Payment Request.* S, which now knows that the transport has been completed, sends a proposal of payment to A, which refers to the transport completion in (4) and the tentative payment in (1). A accepts.

While the scenario illustrates how ordering, transport and payments could be handled in an industrial scenario, there are a few things we want to note.

- No actual payments were issued or executed by the EN used by the three parties, even if several interactions related to money. The purpose of the EN architecture is to facilitate digital changes to the rights and obligations between partners. It does not move any concrete assets in and of itself, even if events in an EN could trigger other systems to perform such functions. However, the signed exchanges resulting from the example interactions should be useful as evidence in a court of law, in the case of any party not meeting its obligations, such as by refusing to pay.
- The EN architecture only relays data that is directly related to the rights and obligations of contractual partners. If other information would be of relevance, such as tracker coordinates or digital twins, that would have to be sent via some other system.
- The example most likely contains too few steps to be practical in a real-world setting. Pick-up, quality checks or other significant interactions could likely be of benefit to also negotiate about. The purpose, however, of the example use case is demonstrate how the technology works and how it *could* be used, not necessarily how it *should* be used.

VI. REQUIREMENTS CONFORMANCE

In Section IV-B, we outline six requirements we believe to be key for facilitating effective contractual cooperation. Here, we briefly introduce and evaluate how (A) our SC implementation of the EN architecture (SC EN), (B) R3 Corda [3], (C) Hyperledger Fabric [4], as well as (D) the original version of Ethereum [5] meet our requirements, after which we highlight the distinguishing characteristics of the evaluated systems. The purpose of this section is to establish that our strategy for implementing the EN architecture does fulfill our requirements for effective contractual cooperation to a degree that is comparable to other available solutions, even if code-as-contracts and distributed consensus algorigms are *not* used. A summary of our evaluations is given in Table 2.

A. SIGNATURE CHAIN EXCHANGE NETWORK (SC EN)

An SC EN is a peer-to-peer system for negotiation that uses signatures and hashes [18] to make negotiation results immutable and useful as proofs.

1) *Negotiable terms.* The EN architecture defines a kind of system that, via its NS component, intrinsically *is* a negotiation system. However, a NS can only fulfill it purpose if there are token definitions that can be used to formulate proposals. As definitions dictate what can be negotiated about, there could, theoretically, be room for negotiating about further definitions, the creation of new contracts, the amendments of existing contracts, as well as contractual exceptions.
2) *Consistent interpretation.* Every EN must provide a DB component, which is intended to maintain definitions of sufficiently rigorous interpretation for negotiation to be practically possible. No particular shape or format is required for any such definitions, even though it may be relevant to support machine-executable validation code and conventional legal prose,[12] as it could enable both automatic proposal verification and provide opportunity for adjudication via existing institutions, respectively.
3) *Interactional privacy.* We understand complete privacy to be the situation in which only those parties that must know a fact do have direct or indirect knowledge of it. While the EN architecture itself does not regulate how proposals are relayed, concrete realizations of the architecture, such as our SC EN implementation, could

[12] While it currently may be advantageous to use traditional legal prose in digital cooperation systems to ensure compatibility with legal institutions, there are tangible benefits to be able to represent it in a directly machine-readable format. It could, for example, allow for machine-assisted contract analysis and verification, which could be used to estimate risks, economical benefits and other relevant factors. *OASIS LegalRuleML* [29] is one example of an already existing document structure, intended to facilitate machine-readable legal contracts.

TABLE 2. An overview of how our EN implementation based on SCs, R3 Corda, Hyperledger Fabric and Ethereum fulfill the requirements we list in Section IV-B, as well as some technical characteristics of each solution. Parentheses are used signify functionality that can only be supported if provided via an external means.

	SC EN	*R3 Corda*	*Hyperledger Fabric*	*Ethereum*
1) *Negotiable terms*	Via NS component	(External negotiation)	(External negotiation)	(External negotiation)
2) *Consistent interpretation*	Via DB component	Code & legal prose	Code (& legal prose)	Code (& legal prose)
3) *Interactional privacy*	Witnesses optional	Notaries optional	Via subgroups	(External anonymization)
4) *Provable acceptances*	Via EL component	Signatures & notaries	Signatures	Signatures & proof-of-work
5) *Effective adjudication*	(External adjudicator)	(External adjudicator)	(External adjudicator)	(External adjudicator)
6) *Trustworthy identification*	Via UR component	Certificates	Certificates	(External identification)
X) *Primary contract model*	Token ownerships	Code-as-contracts	Code-as-contracts	Code-as-contracts
Y) *Consensus scope*	Group-wide	Group-wide	Group-wide	Network-wide
Z) *Performance category*	Higher*	Higher*	Higher* [4]	Lower [10]

∗ We can conclude that these solutions *may* have better performance only from the fact their consensus scopes are significantly smaller.

use transport layer security [30], or any other comparable technology, to conceal data in-transit between parties. Furthermore, since using SCs does not require a distributed consensus algorithm, there is not necessarily any voting procedure during which any details about proposals or exchanges could be seen by third parties.

4) *Provable acceptances.* The SC data structure facilitates provable acceptances by requiring each acceptance to be cryptographically signed both by its proposer and acceptor. Additional parties can also add their signatures or act as witnesses, given a suitable finalization protocol. We describe such a protocol that includes one witness in Section V-B.2. If an acceptance includes hashes of a related earlier acceptance or contractual definitions, no party will be able to deny the history of the acceptance or the commitments associated with it, given that the history and definitions are available to be inspected.
5) *Effective adjudication.* The EN architecture leaves room for using machine-executable tests, which could be used to *prevent* some unsound exchanges from taking place. Neither ENs in general or SC ENs in particular are, however, able to *correct* contractual misbehavior after it has occurred, which we discuss further in Section VII-B. That being said, the SC data structure is designed to be useful for proving that the agreements it records have taken place. That proof, which consists of the cryptographic signatures of well-known stakeholders and hash-based references to contractual definitions, we hope to be useful as evidence in traditional courts of law, which should be able to provide the desired kind of correction.
6) *Trustworthy identification.* The problem of reliably identifying other users is meant to be solved by the UR component of all ENs. However, the EN architecture does not explicitly specify how it is to be facilitated. In our SC EN concept implementation, we use data taken from x.509 certificates [21] to identify parties, which we manually provide to each node. An implementation for production usage would, however, likely need to support distribution of certificates at runtime, handle parties transitioning to new certificates, as well as being able to assess the likelihood of any certificates being compromised.

B. R3 CORDA

Corda [3] is a peer-to-peer system for multi-organizational state machine applications, or *CorDapps*, which are updated via signed [18] and logged function invocations.

1) *Negotiable terms.* Corda is not a negotiation system, it is a system for maintaining and updating replicated state machines via distributed function invocations. In other words, it does not provide any primitives explicitly designed for making or accepting proposals, as defined in this paper. The details of any collaboration must be negotiated outside Corda itself, including how and what can be negotiated about in each given collaboration. That being said, there could likely be room for building a general-purpose negotiation system on top of the primitives Corda does provide. However, providing a programming language and execution runtime is not the same as providing a concrete feature, for which reason we do not consider Corda to directly facilitate negotiable terms here.
2) *Consistent interpretation.* In order to guarantee that code-as-contracts and other machine-readable artifacts are interpreted consistently, Corda comes with its own machine language interpreter. It also leaves room for its transactions to refer to legal prose and other attachments. If such legal prose is formulated in accordance with the norms of some well-established legal tradition, it could provide compatibility with legal institutions.
3) *Interactional privacy.* R3 Corda guarantees privacy by encrypting messages in-transit between parties. Notary pools, when used, are only provided with hashes of state objects, as described briefly in Section III-C, which means that the contents of any considered objects are not accessible to the notaries in any given pool.

4) *Provable acceptances.* In R3 Corda, each cooperational interaction results in the creation of a *transaction*, which contains a list of input state objects, commands to apply to those objects, as well as other details. Each such transaction proves its validity by including at least the signature of its issuer. Transactions can be configured to hold additional signatures as needed via the CorDapps that define them. Pools of notary nodes, which we describe briefly in Section III-C, can also be used as a form of witnesses.
5) *Effective adjudication.* The fact that Corda maintains a replicated state machine means that it leaves plenty of opportunity for programmatically validating states and state changes, which is useful for *preventing* unsound state transitions from taking place. However, Corda does not in and of itself provide any direct means for *correcting* contractual misbehavior that cannot be prevented. Rather, Corda depends on its code-as-contracts, transactions, immutably referenced legal prose, and any other artifacts, to be accepted as evidence by a court of law or other adjudicator.
6) *Trustworthy identification.* To guarantee non-repudiation and to unambiguously associate transactions with the identities of existing legal entities, Corda employs x.509 certificates [21]. Further, it *"assumes [the existence of] an identity infrastructure between the participants in the network but makes no assumption as to its sophistication or mode of operation"* [3]. However, if participating in the global Corda network, a network maintained by the R3 organization, the use of a custom infrastructure created by R3 is mandatory.

C. HYPERLEDGER FABRIC

Fabric [4] is a *"distributed operating system"* for replicating state machines within groups of *peer nodes* using traditional distributed consensus algorithms. Updates take the form of signed [18] and logged function invocations.

1) *Negotiable terms.* Negotiation is not supported unless designed via the code-as-contracts system it provides.
2) *Consistent interpretation.* Code-as-contracts, referred to as *chaincodes*, can be specified in multiple different programming languages, all of which provide well-defined execution or interpretation models. Fabric does not directly provide any legal language support.
3) *Interactional privacy.* While not explicitly mentioned in [4], we see no reason why Fabric would not be able to encrypt messages in-transit between computers. Furthermore, as the consensus groups, or channels, of Fabric can be adjusted to only include directly related parties, it can also be ensured that messages are not seen by parties not immediately concerned.
4) *Provable acceptances.* In Fabric, *transactions*, which encode all interactions, are signed by their issuers before being submitted to the so-called *endorsing peers* of a relevant consensus group, or *channel*. Each such peer that considers the transaction valid signs it. If all signatures mandated by the so-called *endorsement policy* of the channel are added to the transaction, it is eventually appended to the ledger replicated within that channel. The signatures prove that each interaction was accepted by all parties of concern.
5) *Effective adjudication.* Fabric can prevent unsound transactions from taking place by letting users specify validation rules in its code-as-contracts. It does not, on the other hand, directly support embedding legal prose into its transactions, but can support it indirectly via a system such as the one by the *Accord project* [16].
6) *Trustworthy identification.* Fabric provides a so-called *membership service provider* for verifying the identities of all computers interacting with a given system. The identities are connected to legal entities via certificates.

D. ETHEREUM

The original version of Ethereum [5] maintains a *globally* replicated state machine that is updated via signed [18] and logged function invocations. As the system is global and maintained voluntarily, it incentivizes maintenance by hosting a cryptocurrency that is used to reward maintainers and charge for contract execution. The consensus model and currency of Ethereum are comparable to those of Bitcoin [1].

1) *Negotiable terms.* Negotiation is not supported unless designed via the code-as-contracts system it provides.
2) *Consistent interpretation.* Ethereum requires that all contracts be provided in the same byte code format, which guarantees that all network participants interpret the contracts in the exact same way. No legal contract language is provided directly, as we already mentioned.
3) *Interactional privacy.* Neither transactions or contracts are encrypted or concealed in any way. The reason for this is that all network maintainers must be able to verify that each transaction takes the contract it invokes into a valid state. Furthermore, Ethereum does nothing to conceal public keys, which means third parties able to associate public keys with legal entities become able to track all of their activities.
4) *Provable acceptances.* Ethereum, which also represent interactions by *transactions*, only requires that the issuer of each transaction signs it. However, code-as-contracts can be designed to require additional approvals before executing its clauses. Whenever a new transaction is submitted for ledger inclusion, it is associated with a so-called *proof-of-work*, which functions as an indirect proof that the majority of the network considers it to be valid [1], [5]. However, for the reasons we outline and discuss in Sections III-B and VII-B, this form of global validation is more limited than it may initially seem.
5) *Effective adjudication.* While able to prevent some unsound transactions from taking place via its code-as-contracts system, embedding legal prose into contracts is not explicitly supported. Such could,

however, likely be superimposed on the primitives it does provides.

6) *Trustworthy identification.* Participation is allowed without explicit membership registration. Public keys, without any other data, are considered to be valid user identifiers. While no explicit association between public keys and legal entities is made, there is no technical reason such could not be stored outside of Ethereum.

E. DISTINGUISHING CHARACTERISTICS

We hope that reading the requirements analyses in this Section helped establish how similar these four solutions are, in terms of our requirements. In particular, they all use cryptography to prove acceptances, support programmatic verification of interactions, provide, or could theoretically provide, some kind of means of establishing legal identities. Their differences largely come down to their (1) contract languages and (2) consensus scopes, as follows:

1) *Contract language*. The SC EN system provides a negotiation-ownership model, while the others support code-as-contracts. While technically different, both help establish which parties have what rights and obligations at every given instance of a cooperation. The difference between these two approaches are not their theoretical capabilities, but the domain models that must be worked with to make them operational. Our SC EN solution requires formulating what token ownerships imply what rights and obligations, while the other require writing state machine applications in programming languages.
2) *Consensus scope*. Ethereum operates globally, rather than within closed groups of participants as the other three. While a global scope might allow for supporting novel and interesting use cases, it makes it harder to facilitate privacy when interactions are sensitive. Fabric, on the other hand, does allow its channels to include only parties of interest. However, as all parties of concern have to be part of the channel while interacting, it offers less flexibility than R3 Corda and our SC EN, which do not directly superimpose any notion of groups.

VII. DISCUSSION

We have now considered what contractual cooperation is, key requirements for it to be effective, the EN architecture and SC implementation, as well as how our implementation and the other DLTs fulfill our requirements. Here, we discuss the wider implications and weaknesses of our contributions.

A. IS NEGOTIATION A SUFFICIENT ABSTRACTION?

Throughout this paper, we have assumed that *negotiation about token ownerships* to be an adequate abstraction for representing most kinds of contractual cooperation. As the notion of ownership maps well to the notion of owning rights or obligations, we assume our model can replace most uses of conventional contracts, which we, in turn, assume to be an adequate vehicle for most kinds of collaboration. However, the position we take can be argued to require stronger assumptions than the prevailing code-as-contracts model. As programming languages can be *Turing complete*, they can also be able to simulate all other domain models, including the one we present. But if our assumption about the adequacy of our negotiation-based model is sufficiently correct, it offers less complexity than the prevailing model, which could serve to reduce costs by leaving less room for error and by requiring less skills to be utilized.

B. THE PROSPECT OF MACHINE ADJUDICATION

In this paper, we identify distributed consensus algorithms and code-as-contracts as primary obstacles to compatibility with existing adjudicators, such as national courts of law. Those two technologies are, however, what many envision will allow trusted middle-men, among which courts of law are a primary example, to be fully circumvented. However, apart form disrupting current practices, as we already noted, the concrete systems currently using these technologies do have some additional shortcomings. In particular, they are limited in

1) what they can consider as evidence,
2) their power over parties deemed at fault, as well as
3) their ability to take the wider context into account.

In systems like Bitcoin [1] or Ethereum [5], the only unbiased evidence available is signed transactions, the only available punishment is refusing to append transactions to the maintained ledger, and no given data will be considered in terms of its original domain or context. Taken together, this means that these systems are unable to identify or mitigate transactions sent to fraudulent users, for example, even if they are able to detect double-spending, overspending, and so on. Strides of advancement in AI technologies, in areas such as trust assessment, faction mapping and computational ethics would be required in order to make computer systems begin approaching the capabilities of present day institutions. For this reason, we assume that human adjudicators will remain indispensable for many kinds of use cases in the foreseeable future, even though the list of exceptions may grow as DLTs allow for more kinds of middle-men to be circumvented.

C. A MINIMUM-VIABLE INTEGRATION STRATEGY

However useful the EN SC approach we present in this paper may be, we have not considered the practical details of setting up an operational system between multiple stakeholders. If a minimum-viable solution is considered adequate, the following five steps could be a workable starting-point.

1) Each party of the collaboration in question installs and configures an EN SC software on a server it manages.
2) Cryptographic certificates are generated by each party, and uploaded to the UR of every server.
3) Legal documents are scanned, hashed and uploaded to the DB of each server.

4) Key interactions defined by the legal documents are identified, and tokens referring to them are specified.
5) A new legal document is signed by each participant, containing their cryptographic certificates.

Essentially, finalized EN exchanges are used as digital receipts referring to the terms of existing paper contracts. If assuming that the EN SC software provides a graphical user interface, no further technical integration would be strictly required as humans would be able to manage all interactions. Each party would be able to independently automate any of the interactions in which it takes part.

D. THE ECONOMICS OF SYSTEM PARTICIPATION

It seems to us that participating in a digital collaboration system always will require (1) installing, configuring and connecting a node to a network; (2) reliably identifying the user identities of any relevant counter-parties; as well as (3) formulating and distributing contractual definitions. We think that the complexity and costs associated with these three steps will have significant impact on the real-world flexibility and utility of these systems, even though that complexity and cost may be hard to quantify. We argue that the lower complexity of our ownership-based domain model, compared to a Turing complete programming language, could yield lower costs for formulating contracts, which would make systems using the model more flexible in terms of being able to change and enter into new collaborations.

E. CONSENSUS PERFORMANCE AND DISRUPTION

In Section I, we claimed that the performance of distributed consensus algorithms could be a major disruptor. While no benchmarks are presented in this paper, we want to note that the theoretical performance of the implementation we propose in Section V-B should be excellent, as it assumes consensus will be sought primarily between pairs of nodes.

F. THE LIMITED QUALITY OF DLT WHITE-PAPERS

It seems to us as if the tradition within the distributed ledger community is to present technical findings and solutions in white-papers. Apart from the textual and factual issues that would not have been present in case of serious peer-reviews, these white-papers often seem to be vague about technical limitations, perhaps for marketing reasons, and be updated infrequently. Consequently, we cannot guarantee the accuracy of our findings to the extent that otherwise would have been possible. Despite this, we see no significant risks of our most important claims, such as those about the adequacy of our negotiation-based model, being compromised by any technical details about these solutions being incorrect.

G. RIGHTS AND OBLIGATIONS AS STATE MACHINES

Even though we present the Exchange Network concept as facilitating *negotiation about token ownership*, via the use of an abstract message protocol, there is nothing preventing that a computational model be superimposed on that protocol. Concretely, tokens refer to *types*, which in turn refer to *tests* and *terms*, as described in Section V. One could regard a test as an invariant of a type system, a term as a function body, a type as a full function declaration, and a token as a function invocation. As tokens have room for arbitrary data items, they could even be said to include function arguments. If taking this perspective, finalized negotiations result in functions updating a state machine of rights and obligations. Even though this does not imply Turing-completeness, it does suggest that there could be interesting mathematical parallels between the code-as-contracts model and the one we propose. Formally defining such a state machine and comparing it to the state-of-the-art is left as a topic for future research.

VIII. CONCLUSION

In this paper, we make the case that code-as-contracts and distributed consensus algorithms disrupt common business practices. We also claim that those disruptors can be replaced by a system of negotiated token exchanges and non-mediated message passing. By substituting the prevailing finite state machine code-as-contracts model with one of negotiations about ownership exchanges, we change the primary concern of the model from function invocations and state transitions to exchanges of rights and obligations. By using cryptographic signatures and hash pointers, akin to R3 Corda [3], we ensure messages sent directly between two peers can be proved to be authentic later to third parties.

We believe existing professionals, such as procurement engineers, legal experts and adjudicators, will be able to fruitfully apply the kind of technology we propose with little training, given that the right kind of supporting software will exist. Writing software is far removed from what most relevant kinds of professionals are accustomed to, and the use of voting to ratify interactions is rare in conventional kinds of collaboration. However, digital signatures, ownership statements, as well as the other primitives our system design provides, should be perceived as more familiar and, therefore, require less training, as well as requiring fewer adaptations to existing business norms and practices. If the assumptions we make are correct, our approach lowers the barriers to adoption of distributed ledger technologies for businesses, legal institutions and others in comparison to state-of-the-art solutions, such as Hyperledger Fabric [4] or R3 Corda [3].

That being said, there might be compelling use cases that cannot be facilitated by our approach. For example, solutions such as Ethereum [5] are able to facilitate code-controlled agents via a public and global process reminiscent of voting, which can be used to circumvent traditional third parties in certain situations. Our current understanding is that nothing similar could be achieved with the system design we propose, at least without extending it to also support defining the behavior of and facilitating such agents. However, the practical utility of such agents has proven very limited, as we discuss in Section VII-B, which means that they cannot replace many interesting third parties to an adequate degree, such as inspection firms or courts of law.

In Section IV-A, we characterize cooperation as *being* the process of collaborating parties continuously renegotiating their current sets of rights and obligations, while never being absolutely sure about the aims and incentives of their counterparties. If nothing else, we believe this paper should establish that digital cooperation systems must be able to represent the contentious nature of this process to remain useful over time.

ACKNOWLEDGMENT

We would like to thank Richard Hedman at Volvo Group for his insights regarding industrial supply chains. We would also like to thank Caroline Berg von Linde, Johan Hörmark, Christian Lagerkvist and Jamie Walters at SEB for helping us better understand the utility of trusted middle-men.

REFERENCES

[1] S. Nakamoto. (2008). *Bitcoin: A Peer-To-Peer Electronic Cash System*. Accessed: Feb. 8, 2019. [Online]. Available: http://bitcoin.org/bitcoin.pdf

[2] M. E. Peck, "Blockchains: How they work and why they'll change the world," *IEEE Spectr.*, vol. 54, no. 10, pp. 26–35, Oct. 2017, doi: 10.1109/mspec.2017.8048836.

[3] M. Hearn. (2016). *Corda: A Distributed Ledger*. Accessed: Feb. 22, 2119. [Online]. Available: https://www.corda.net/content/corda-platform-whitepaper.pdf

[4] E. Androulaki *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proc. 13th EuroSys Conf.-EuroSys*, 2018, pp. 30:1–30:15, doi: 10.1145/3190508.3190538.

[5] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow*, vol. 151, pp. 1–32, Apr. 2014, [Online]. Available: http://gavwood.com/paper.pdf

[6] S. S. Arumugam, V. Umashankar, N. C. Narendra, R. Badrinath, A. P. Mujumdar, J. Holler, and A. Hernandez, "IOT enabled smart logistics using smart contracts," in *Proc. 8th Int. Conf. Logistics, Informat. Service Sci. (LISS)*, Aug. 2018, pp. 1–6, doi: 10.1109/liss.2018.8593220.

[7] L. Chen, W.-K. Lee, C.-C. Chang, K.-K.-R. Choo, and N. Zhang, "Blockchain based searchable encryption for electronic health record sharing," *Future Generation Comput. Syst.*, vol. 95, pp. 420–429, Jun. 2019, doi: 10.1016/j.future.2019.01.018.

[8] U. M. Ramya, P. Sindhuja, B. B. Dharani, S. S. M. V. Golla, and R. A. Atsaya, "Reducing forgery in land registry system using blockchain technology," in *Advanced Informatics for Computing Research*. Singapore: Springer, 2018, pp. 725–734, doi: 10.1007/978-981-13-3140-4_65.

[9] F. Casino, T. K. Dasaklis, and C. Patsakis, "A systematic literature review of blockchain-based applications: Current status, classification and open issues," *Telematics Informat.*, vol. 36, pp. 55–81, Mar. 2019, doi: 10.1016/j.tele.2018.11.006.

[10] A. Chauhan, O. P. Malviya, M. Verma, and T. S. Mor, "Blockchain and Scalability," in *Proc. IEEE Int. Conf. Softw. Quality, Rel. Secur. Companion (QRS-C)*, Jul. 2018, pp. 122–128, doi: 10.1109/qrs-c.2018.00034.

[11] M. Giancaspro, "Is a 'smart contract' really a smart idea? Insights from a legal perspective," *Comput. Law Secur. Rev.*, vol. 33, no. 6, pp. 825–835, Dec. 2017, doi: 10.1016/j.clsr.2017.05.007.

[12] E. Palm, O. Schelén, U. Bodin, and R. Hedman, "The exchange network: An architecture for the negotiation of non-repudiable token exchanges," in *Proc. IEEE 17th Int. Conf. Ind. Inform. (INDIN)*, to be published. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:ltu:diva-74043

[13] R. Klomp and A. Bracciali, "On symbolic verification of bitcoin's script language," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Cham, Switzerland: Springer, 2018, pp. 38–56, 10.1007/978-3-030-00305-0_3.

[14] J. D. Bruce. (2014). *The Mini-Blockchain Scheme*. Accessed: Mar. 20, 2019. [Online]. Available: http://cryptonite.info/files/mbc-scheme-rev3.pdf

[15] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, Sep. 1997, doi: 10.5210/fm.v2i9.548.

[16] *The Smart Legal Contract Stack, Accord Project LLC*. Accessed: Feb. 25, 2019. [Online]. Available: https://www.accordproject.org

[17] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016, doi: 10.1109/access.2016.2566339.

[18] A. Salomaa, *Public-Key Cryptography* (Texts in Theoretical Computer Science), 2nd ed. Berlin, Germany: Springer, 1996, doi: 10.1007/978-3-662-03269-5.

[19] C. Bell, *Introducing the MySQL 8 Document Store*. New York, NY, USA: Apress, 2018, doi: 10.1007/978-1-4842-2725-1.

[20] *FIPA Communicative act Library Specification*, document FIPA 00037, Foundation for Intelligent Physical Agents, May 2019. [Online]. Available: http://www.fipa.org/specs/fipa00037

[21] D. Cooper, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, document RFC 5280, Internet Request for Comments, May 2008, doi: 10.17487/RFC5280.

[22] R. Fielding and J. Reschke, *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*, documnet RFC 7230, Internet Request for Comments, 2014, doi: 10.17487/RFC7230.

[23] I. Fette and A. Melnikov. *The WebSocket Protocol*, document RFC 6455, Internet Request for Comments, Dec. 2011, doi: 10.17487/RFC6455.

[24] G. Bierman, M. Torgersen, and M. Abadi, "Understanding typescript," in *Proc. Eur. Conf. Object-Oriented Program.* Berlin, Germany: Springer, 2014, pp. 257–281, doi: 10.1007/978-3-662-44202-9_11.

[25] A. Wirfs-Brock, *ECMAScript 2015 Language Specification*, document ECMA-262, ECMA International, 2015. [Online]. Available: http://www.ecma-international.org/ecma-262/6.0

[26] S. Tilkov and S. Vinoski, "Node.js: Using javascript to build high-performance network programs," *IEEE Internet Comput.*, vol. 14, no. 6, pp. 80–83, Nov. 2010, doi: 10.1109/mic.2010.145.

[27] S. Faulkner, A Eicholz, T. Leithead, A Danilo, and S. Moon. (2017). *HTML 5.2, W3C Working Draft*. [Online]. Available: https://www.w3.org/TR/2018/WD-html53-20180809

[28] T. Atkins and S. Sapin, *CSS Syntax Module Level 3*, document CR-css-syntax-3-20140220, W3C Candidate Recommendation, Cambridge, MA, USA, Feb. 2014. [Online]. Available: http://www.w3.org/TR/2014/CR-css-syntax-3-20140220

[29] T. Athan, M. Palmirani, A. Paschke, A. Wyner, and G. Governatori, "LegalRuleML: Design principles and foundations," in *Reasoning Web. Web Logic Rules*. Cham, Switzerland: Springer, 2015, pp. 151–188, doi: 10.1007/978-3-319-21768-0_6.

[30] E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.3*, document RFC 8446, Internet Request for Comments, Dec. 2018, doi: 10.17487/RFC8446.

EMANUEL PALM received the M.Sc. degree in mobile systems from the Luleå University of Technology, where he is currently pursuing the Ph.D. degree. He has worked as an IT Consultant, web developer and mobile application developer. His master's thesis was on understanding the implications and impact of blockchain transaction pruning, which lead to his current work on digital negotiation, ownership and contracts, and their application within the industrial IoT.

ULF BODIN received the Ph.D. degree in computer communications from the Luleå University of Technology, Sweden, in 2003. He holds a position as an Associate Professor at the Luleå University of Technology, where he is conducting research on the industrial IoT, distributed system of systems, computer communications, distributed ledgers, and applied machine learning. His experience includes working in the software industry and in ETSI and several other standardization organizations.

OLOV SCHELÉN received the Ph.D. degree in computer networking from the Luleå University of Technology. He is currently an Associate Professor with the Luleå University of Technology and a CEO with Xarepo AB. Thereafter, he has more than 20 years of experience from industry and academia. His research interests include mobile and distributed systems, software orchestration, computer networking, artificial intelligence, and blockchain.