On and Off-Blockchain Enforcement Of Smart Contracts

Carlos Molina-Jimenez[1], Ellis Solaiman[2] ✉, Ioannis Sfyrakis[2], Irene Ng[3], and Jon Crowcroft[1]

[1] Computer Laboratory, University of Cambridge, UK
{carlos.molina,jon.crowcroft}@cl.cam.ac.uk
[2] School of Computing, Newcastle University, UK
{ellis.solaiman,ioannis.sfyrakis}@newcastle.ac.uk
[3] Hat Community Foundation, Cambridge, UK
irene.ng@hatcommunity.org

Abstract. Emerging blockchain technology is a promising platform for implementing smart contracts. But there is a large class of applications, where blockchain is inadequate due to performance, scalability, and consistency requirements, and also due to language expressiveness and cost issues that are hard to solve. In this paper we explain that in some situations a centralised approach that does not rely on blockchain is a better alternative due to its simplicity, scalability, and performance. We suggest that in applications where decentralisation and transparency are essential, developers can advantageously combine the two approaches into hybrid solutions where some operations are enforced by enforcers deployed on–blockchains and the rest by enforcers deployed on trusted third parties.

Keywords: Smart Contracts, Blockchain, Monitoring, Enforcement, On chain, Off chain, IoT, Privacy, Trust.

1 Introduction

This paper focuses on scenarios where two or more parties interact with each other to conduct business over the Internet. Typical scenarios involve consumers and providers where the latter sell tangible items or computing services to the former. A specific example is the selling of personal data collected from IoT sensors or social media applications to data consumers. We assume the business parties involved are reluctant to trust each other unguardedly, that is; without software mechanisms that assure 1) parties act according to some agreed upon rules, and 2) performed actions are indelibly recorded on means that make them undeniable and examinable, for example, to determine the sequence of actions that led to an unexpected outcome and subsequent dispute.

In conventional business, the mechanisms normally used in these situations are business contracts supported by *ledgers*. The contract stipulates what actions the parties are expected to execute, while the ledger is used to record the history of the actions that have been executed. It is widely accepted that equivalent mechanisms are also needed in electronic business. An emerging solution that is currently being explored to address this

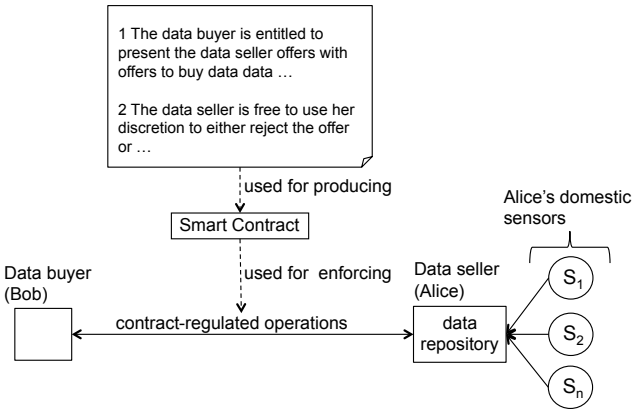

Fig. 1. Data trading regulated by a smart contract.

question is **smart contracts** built on the basis of blockchain technologies [1] [2]. Examples of such technologies are Bitcoin [3], Ethereum [4] and Hyperledger [5]. However, blockchain-based smart contracts are only at their initial research stage, and plagued with questions about their scalability, performance, transaction costs and other questions that emerge from their decentralised nature.

This article makes the following contributions to help clarify some of these issues. i) We explain that there are different approaches to implement smart contracts ranging from centralised to decentralised. ii) We explain the advantages and disadvantages of these approaches and argue that their suitability in solving the problem depends on the particularities of the application, the assumptions made about the application, and the facilities offered by the blockchain technology available. iii) We argue that there is a large class of applications that can benefit from a hybrid solution.

The remainder of this article is organised as follows: Section 2 presents a contract example to motivate the use of smart contracts. In Section 3, we introduce smart contracts and describe the difference between the centralised and decentralised variations. Section 4 discusses implementation alternatives of smart contracts. Section 5 places our work within past and current contexts. In Section 6, we present some concluding remarks and raise questions that in our view, need research attention.

2 Motivating scenario

An illustrative example of a contractually regulated IoT application of our research interest is shown in Fig. 1. Alice is a person in possession of personal data that she would like to sell and as such she plays the role of a *Data seller*. The *Data Buyer* (represented by Bob) is a company interested in buying data from Alice. Alice gathers her data from different sources, such as her social network activities, body sensors and domestic sensors, as envisioned in [6]. For simplicity and to frame the discussion, we assume that Alice is trading only her data collected from her domestic sensors. We assume that Alice stores her data in a personal repository, perhaps located in the cloud.

Like in the "Hat" project [7], we assume Alice is the absolute owner of the data and that she is entitled to negotiate with potential buyers how to trade her data, i.e., to whom to sell it to, when, and under which conditions. The negotiation process can be as sophisticated as needed. Since this issue falls outside of the scope of this paper, we consider only a simple *accept or reject the offer as it is* negotiation process. An example of contractual clauses that Alice and Bob can use to regulate their data trading follows:

1. *The buyer (Bob) is entitled to present the data seller (Alice) with* ***offers to buy data*** *collected from Alice's domestic sensors.*
2. *The data seller is free to use her discretion to either* ***reject the offer*** *or* ***accept the offer*** *as it is.*
   (a) *The data seller is expected to* ***send a notification of offer acceptance*** *within 36 hrs of receiving the offer, when she decides to accept it.*
   (b) *Failure to send a notification will be considered as offer rejection.*
3. *The data buyer is obliged to* ***send the payment*** *to the data seller within 24 hrs of receiving the notification of acceptance.*
   (a) *Failure to meet his obligation will result in an abnormal termination of the agreement to be sorted out off line.*
4. *The data seller is obliged to* ***send a notification of payment acceptance*** *to the data buyer within 24 hrs of collecting the payment.*
   (a) *Failure to meet his obligation will result in an abnormal termination of the agreement to be sorted out off line.*
5. *The data seller is obliged to* ***make the data available*** *to the data buyer within 24 hrs of collecting the payment and* ***maintain the data repository accessible*** *during the following seven days.*
6. *The Data buyer is entitled to* ***place data requests*** *against the data seller repository without exceeding 24 data requests per day.*
7. *The data buyer is entitled to* ***close the repository*** *upon expiration of the seven day period.*
8. *This agreement will be considered successfully complete when the seven day period expires.*

The clauses include several contractual operations that we have highlighted in bold such as *offer to buy data*, *reject the offer*, *accept the offer*, *send a notification of offer acceptance*, *send payment*, etc. Though the clauses are relatively simple, they are realistic enough to illustrate our arguments.

3 Smart contracts: background

A smart contract is an event–condition–action stateful computer program, executed between two or more parties that are reluctant to trust each other unguardedly. It can be regarded as Finite State Machine (FSM) that keeps a state that models the development (from initiation to completion) of a shared activity. For instance, in [8] [9], the state is used for modeling changes in rights, obligations and prohibitions as they are fulfilled or violated by the parties.

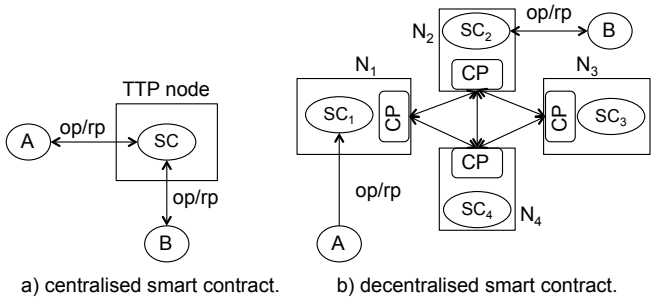

Fig. 2. Centralised and decentralised implementation of a smart contract.

Research on executable contracts can be traced back to the mid 80s and early 90s [10,11]. In 1997, Szabo used the term smart contract [12] to refer to contracts that can be converted into computer code and executed. However, commercial interest in smart contracts emerged only in 2008 motivated by the publication of Satoshi's Bitcoin paper [13] that inspired the development of cryptocurrencies, smart contracts and other distributed applications. Satoshi departed from the centralised approach taken in previous research and demonstrated how smart contracts can be decentralised.

3.1 Centralised and decentralised smart contracts

Depending on the number of instances (copies) of the smart contract deployed to monitor and enforce the contract we distinguish between centralised and decentralised (distributed) approaches (Fig. 2). In the figure, *A* and *B* are business partners, for example, Alice and Bob of our contract example of Section 2. *SC* is the corresponding smart contract. *op* stands for operation executed against *SC*, *rp* is the corresponding response. *TTP node* is a node under the control of a Trusted Third Party. $N_1, \ldots, N_4$ are untrusted nodes. *CP* stands for Consensus Protocol. As shown in Fig. 2–a), a contract can be implemented as a centralised application that uses a single instance of the smart contract (*SC*) running in the TTP node. Besides the disadvantages that a TTP introduces (single point of failure, trust placed on the TTP, etc.) this approach is comparatively simpler that the decentralised approach. The decentralised approach relies on a set of untrusted nodes instead of a single TTP that are used for running several identical instances (shown as $SC_1, \ldots, SC_4$) of the smart contract. In this approach, *A* and *B* are free to place their operation against any of the instances. The price that the decentralised approach pays for getting rid of the TTP is that the untrusted nodes must run a consensus protocol to verify that a given operation has been executed correctly, and to keep the states of $SC_1, \ldots, SC_4$ identical. Depending on the protocol used, its computational, communication and performance degradation cost might be unbearable [14] or its consistency guarantees inadequate [15] to the extent of rendering the decentralised approach unsuitable.

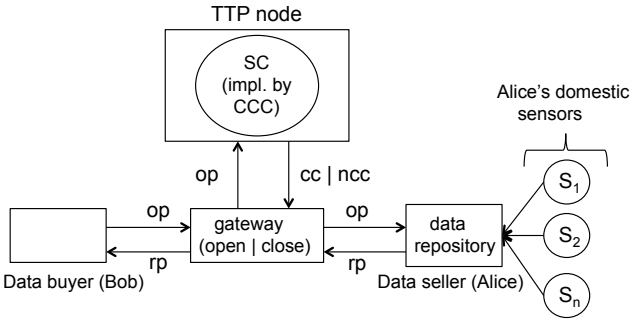

Fig. 3. Centralised smart contract.

4 Implementation alternatives

We will take the example of Section 2 and highlight the advantages and disadvantages of three implementation alternatives.

4.1 Centralised implementation

A centralised implementation is shown in Fig. 3. The role of the *SC* is played by the CCC (Contract Compliance Checker) developed at the University of Newcastle. We use CCC and SC synonymously in this section. The CCC is a FSM written in Java that accepts contractual clauses encoded as business rules written in the Drools language [8]. The state of the FSM is altered by the execution of contractual operations (*op*) initiated by the business partners, such as *offer to buy data*, and *send the payment*. The FSM running within the CCC keeps track of the state of the business process executed between Bob and Alice, and on this basis it determines if a given operation is contract compliant (*cc*) or non contract compliant (*ncc*). The CCC is used to control the *gateway* that grants access to Alice's data. For example, when Bob wishes to access Alice's data, he i) issues the corresponding operation against the gateway, ii) the gateway forwards the operation to the CCC, iii) the CCC evaluates the operation in accordance with its business rules that encode the contractual clauses and responds with either *cc* or *ncc* to open or close the gateway, respectively, iv) the opening of the gateway allows Bob's operation to reach the data repository and retrieve the response (*rp*) that travels to Bob. Note that, to keep the figure simple, the arrows show only the direction followed by operations initiated by Bob.

It is worth elaborating the following points. Observe that in the architecture, all the operations are presented to the *SC* for evaluation. The operation rate is not a problem because the architecture involves only a single instance of the *SC*, i.e., there is no need to run consensus protocols. Likewise, the contract clauses, which are encoded in the Drools languages, are executed by a FSM implemented in Java. This means that we have a Turing complete programming environment that allows us to encode and implement clauses of arbitrary complexity. Unfortunately, the centralised approach introduces several drawbacks. For example, the contracting parties need to trust the TTP

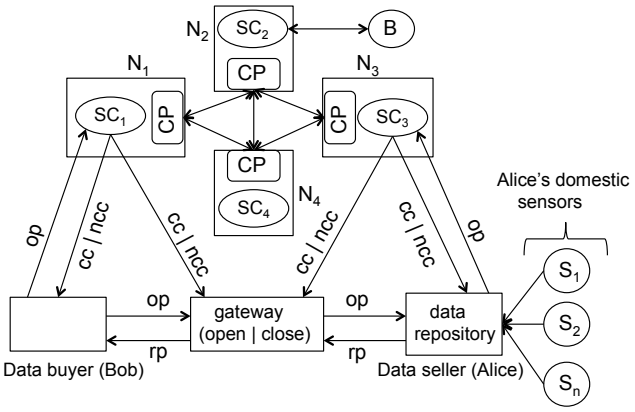

Fig. 4. Decentralised smart contract.

to collect undeniable and indelible records of the actions executed by the contracting parties and make them available upon request to parties that are entitled to see them, say to sort out disputes. At the technical level, the TTP is a single point of failure. Another issue is that the execution of the payment operation is centralised. We assume a conventional card payment mediated by a bank as opposed to cryptocurrency payment.

4.2 Decentralised implementation

A decentralised architecture is shown in Fig. 4. Four instances of the smart contract ($SC_1, \ldots, SC_4$) are deployed in four nodes $N_1, \ldots, N_4$ (one each) of a blockchain platform. Each operation initiated by a business partner is executed against the contract; the contract determines if the operation is contract compliant (*cc*) or non contract compliant (*ncc*) and responds to both business partners accordingly. The response is also sent to the *gateway* to open or close it, accordingly.

To keep the figure simple, we show only the communication lines between the *Data buyer*, SC_1 and the *gateway*; and between the *Data seller*, SC_3 and the *gateway*. Yet we assume that a given operation can be presented to any of the four instances of the smart contract and that any of them can respond to the business partners and the *gateway*.

The salient feature of the decentralised implementation is the replication of the smart contract, consequently, there is no dependency on a single party. The cost to pay for this benefit is the execution of the consensus protocol among the instances which can significantly impact the performance of the smart contract in terms of number of operations (called transactions in blockchain terminology) per second that it can analyse, and the response time to complete a transaction. For example, Bitcoin, a public blockchain that uses a Proof of Work (PoW) consensus algorithm, can only process about 7 transactions per second. Another problem with Bitcoin is its consistency latency: its PoW algorithm offers only eventual consistency that might take Bitcoin about an hour (or longer) to approve and indelibly include a transaction in its blockchain [16]. Ethereum operating under PoW consensus suffer from similar drawbacks. Permissioned

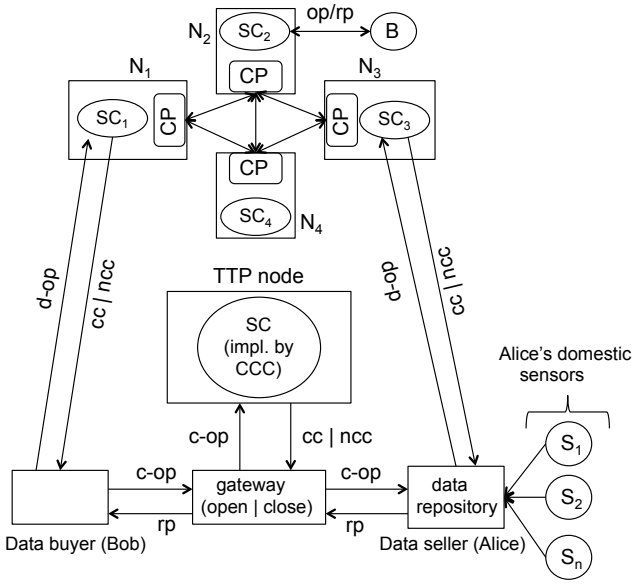

Fig. 5. Hybrid smart contract.

blockchains like Hyperledger rely on lighter consensus algorithms such as Proof of State (PoS). However, applications where eventual consistency is unsafe, demand strong consistency [15]. Strong consistency can only be delivered by communication intensive consensus protocols such as Byzantine Fault Tolerant protocols. Unfortunately, these protocols suffer from scalability issues [14]. Some smart contract applications (for example, applications that require instantaneous payment or the delivery of real time data) fall within this category. Another issue that impacts decentralised approaches that rely on public blockchains is the transaction fee incurred by each operation analysed by the smart contract. In this order, it would be insensible to take a decentralised implementation approach for the contract example of Section 2 if the data buyer was to place a large number of operations to retrieve small pieces of data under stringent time constraints.

4.3 Hybrid implementation

Fig. 5 shows the architecture of a hybrid implementation. It combines features from the centralised and decentralised approaches discussed, respectively, in Sections 4.1 and 4.2. We separate the contractual operations into two classes: decentralised operations (*d–op*) that need blockchain support and operations that can be executed in a centralised fashion (*c–op*). *d–op* operations are encoded using the decentralised approach and enforced by the instances ($SC_1, \ldots, SC_4$) whereas operation of the *c–op* category are encoded using the centralised approach and enforced by the CCC.

The designer separates the contractual operation into *d–op* and *c–op* on the basis of several criteria. As examples, we can mention some key parameters related to the

blockchain technology. The list is meant to be illustrative rather then exhaustive. Complementary advise is discussed in [17,18] where they take into account privacy concerns along with computation and data storage costs.

One decision criterion is the expressiveness of the language used for writing the contract. For instance, if the blockchain does not offer a Turing–complete language, the implementers needs to keep the *d–op* category simple. Bitcoin for example, offers only a stack–based opcode scripting language that does not support loops or flow control structures. In contrast, in a blockchain platform like Ethereum that offers a Turing–complete language the designer can afford to pass as much complexity to the decentralised part of the figure as she wishes to. Another decision criterion is the transaction fee which is an issue in private blockchains like Bitcoin and Ethereum but not in Hyperledger [19] when it is operated as a permissioned blockchain. For example, Bitcoin and Ethereum have already experienced average transaction fees of 54.90 and 4.15 USD, respectively [20]. Another central parameter to take into account is the performance of the blockchain, for example, the number of transactions per second and consistency requirements as explained in Section 4.2. Operations that demand strong consistency would be good candidates to be implemented as *c–op*. The performance of the blockchain is especially relevant to IoT applications where transactions must be automatically monitored to ensure that they perform under strict Quality of Service requirements. For example one could easily imagine an additional clause being added to the contract in Section 2 requiring the repository to process each request for data at a particular rate that would be too fast to be monitored using a smart contract deployed on a blockchain. In such a scenario, a centralised smart contract would be more logical, whereas the blockchain would be used to record important milestone events such as the sending and receipt of payments for received data. We envision that the centralised and decentralised integration can be operated in several ways, including the following:

Indelible blockchain–based log We can operate the blockchain–based part of Fig. 5 as a passive log that records events that the parties consider worth duplicating in the blockchain as well as in the TTP node. By passive we mean that $SC_1, \ldots, SC_4$ are not involved in enforcing activities—this is entirely the responsibility of the CCC. This arrangement is useful when one or more of the contracting parties is reluctant to trust the TTP blindly, say because it is deployed within the buyer's premises.

In this arrangement, the *d–op* set will include operations aimed at creating additional records while *c–op* will include all the contractual operations like in 4.1. The CCC and $SC_1, \ldots, SC_4$ operate independently from each other.

Cryptocurrency–based payment channel The data buyer of the example of Section 2 can take advantage of payment services offered by a public blockchain (for example, Bitcoin) and use the top part of Fig. 5 to pay in satoshis. This approach is recommended only when the payment operation is significantly larger than the transaction fees and is not repetitive. In this arrangement, the *d–op* set will include only the *send the payment* operation stipulated in clause 3. In this arrangement, the CCC requires the assistance of the smart contract running in the blockchain ($SC_1, \ldots, SC_4$) only to verify that the data buyer has fulfilled his obligation to pay. For instance, the data buyer application

can submit his payment through Bitcoin, wait for the confirmation of his transaction, collect the evidence and submit it to the CCC.

Off-blockchain execution of operations In this arrangement the CCC running in the TTP node is used as an off the blockchain channel. The designer places in the *d–op* set only the contractual operations that need decentralised treatment and leaves the remaining in the *c–op*. Naturally, operations that cannot be executed in the decentralised blockchain because of the issues discussed in Section 4.2 need to be included in *c–op* set. A good candidate operation to place in the *d–op* set is *send the payment* (see Section 4.3). Another candidate is *close the repository* when the data seller wishes to generate indelible records about the closing time of her repository and completion of the contract. The remaining operations can be cheaply and efficiently enforced by the CCC, the inclusion of *place data requests* (clause 6), in the *c–op* set is highly desirable because its recurrence would incur high accumulative transaction fees.

It is worth clarifying that there are some similarities between the deployment shown in Fig.5 and the lightning channels for executing off–blockchain payments in Bitcoin [21]. However, observe that in lighting networks the aim is to create channels for conducting micro–payment operations off the blockchain to save on transaction fees. In contrast, in Fig. 5 we use the CCC (a complete contractual enforcing tool) to execute most of the contractual operations off–blockchain. Operations from both sets are independently converted to smart contracts and enforced at run time.

5 Related work

An extended version of this paper can be found here [22]. Research on smart contracts was pioneered by Minsky in the mid 80s [10] and followed by Marshall [11]. Though some of the contract tools exhibit some decentralised features [23], those systems took mainly centralised approaches. Within this category falls [24] and [25]. To the same category belongs the model for enforcing contractual agreements suggested by IBM [26] and the Heimdhal engine [27] aimed at monitoring state obligations (see clause 5 of the contract example, *maintain the data repository accessible*). Directly related to our work is the Contract Compliant Checker reported in [8] [9] which also took a centralised approach to gain in simplicity at the expense of suffering from all the drawbacks that TTPs inevitably introduce. Smart contracts were known as executable contracts or electronic contracts in [28] [29] [30], where the important issues of smart contract representation and verification were discussed. A pioneering implementation of a decentralised contract enforcer is discussed in [31]. The central idea is the use of a distributed middleware that is responsible for keeping indelible records of the operations executed by each party. The middleware (called Business to Business Objects [32]) is in essence an indelible ledger similar in functionality to the hyperledger used by current blockchains.

The publication of the Bitcoin paper [13] motivated the development of several platforms for supporting the implementation of decentralised smart contracts. Platforms in [3], [4], and [5] are some of the most representative. A good summary of the features offered by these and other platforms can be found in [2]. Though they differ on language expression power, fees and other features discussed in Section 4.2 they are convenient

for implementing decentralised smart contracts. The hybrid approach that we suggest addresses problems that neither the centralised or decentralised approach can address separately and was inspired by the off–blockchain payment channel discussed in [21,3]. Similar to our work also is Ekiden, a system for combining blockchains with Trusted Execution Environments (TEEs) [33]. The authors report significant performance improvements however they do not discuss the challenges of testing and verification hybrid smart contracts. The concept of logic–based smart contracts discussed in [34] has some similarities with our hybrid approach. They suggest the use of logic–based languages in the implementation of smart contracts capable of performing on–chain and off–chain inference. The difficulty with this approach is lack of support of logic–based languages in current blockchain technologies. In our work, we rely on the native languages offered by the blockchain platforms, for example, Ethereum's Solidity.

6 Conclusions and Future Research Directions

The central aim of this paper is to argue that conventional business contracts can be automated (at least partially) and that depending on several factors, the centralised approach suits some applications but others demand decentralised implementations or even hybrid implementations. We are only starting to explore hybrid implementation of smart contracts, yet on the basis of the study of the APIs (JSON–RPC) that Bitcoin, Ethereum and Hyperledger offer, the idea seems implementable [35]. Also, it is of practical interest as it would offer a pragmatic answer to the scalability problems that afflict current blockchain platforms. This approach opens several research questions.

An important issue is the interaction between the centralised (CCC) and decentralised components. In Fig. 5 they cannot communicate directly. We are currently working on a version of the CCC that can be deployed as a micro–service capable of interacting with the JSON-RPC Client API that blockchain technologies offer. Precisely, we are investigating how the hybrid architecture can be realised using the Ethereum blockchain and a CCC implemented as a decentralised application (DApp) [36]. The relationship (directly or indirectly) between the CCC and the blockchain raises several questions that need further investigation. They can interact directly, indirectly, tightly or loosely. Fig. 5 suggests the latter where, for example, the CCC can fail and recover while the *send the payment* operation is taking place through the block–chain based smart contract (recall in Bitcoin it might take longer that 24 hrs to complete a transaction). However, in some applications a tight relationship might be desirable to hold or divert the progress of one of the contracts when its counterpart experiences an exception or fails. Therefore it is important to develop an understanding on how to separate the contractual operations into *c–op* and *d–op* in a manner that the two contracts collaborate instead of conflicting with each other. For contracts with scores of clauses, this issue might require the assistance of model–checking tools to ensure that the whole contractual clauses are consistent and that the two sets do not conflict with each other [37,38].

Another issue is the language for writing the contract. It is arguably accepted that declarative languages (rule based languages in particular) are more convenient than imperative languages to encode contractual clauses. This feature is enjoyed by the CCC. However, current blockchain platforms support only imperative languages (for example

Ethereum's Solidity). This means that in our hybrid approach the contract will be written in two different languages which will make their interaction less intuitive. Therefore ideally blockchain platforms should support declarative languages, or alternatively developers should be offered a declarative language that can be automatically translated to languages like Solidity or Drools as needed.

Acknowledgements Carlos Molina-Jimenez is collaborating with the HAT Community Foundation under support of Grant RG90413 NRAG/536. Ioannis Sfykaris was partly supported by the EU Horizon 2020 project PrismaCloud (https://prismacloud.eu) under GA No. 644962.

References

1. K. O'Hara, "Smart contracts– dumb idea," *IEEE Internet Computing*, vol. 21, no. 2, 2017.
2. M. Bartoletti and L. Pompianu, "An empirical analysis of smart contracts: platforms, applications, and design patterns," https://arxiv.org/pdf/1703.06322.pdf, 2017.
3. A. Antonopoulos, *Mastering Bitcoin*, 2nd ed. O'Reilly, 2017.
4. Ethereum, "A next-generation smart contract and decentralized application platform," https://github.com/ethereum/wiki/wiki/White-Paper, Visited 23 Oct 2017 2017.
5. The Linux Foundation, "Hyperledger," www.hyperledger.org, Visited Nov 2017 2017.
6. "HATDex: Rumpel Platform," http://www.hatdex.org/rumpel-platform/, 2018.
7. "Hat: Hub-of-all-things," http://hubofallthings.com/home/, visited: 10 Feb 2016.
8. C. Molina-Jimenez, S. Shrivastava, and M. Strano, "A model for checking contractual compliance of business interactions," *IEEE Trans. on Service Computing*, vol. PP, no. 99, 2011.
9. E. Solaiman, I. Sfyrakis, and C. Molina-Jimenez, "A state aware model and architecture for the monitoring and enforcement of electronic contracts," in *Proc. IEEE 18th Conference on Business Informatics (CBI'2016)*, 2016.
10. N. H. Minsky and A. D. Lockman, "Ensuring integrity by adding obligations to privileges," in *Proc. 8th Int'l Conf. on Software Engineering*, 1985, pp. 92–102.
11. L. F. Marshall, "Representing management policy using contract objects," in *Proc. IEEE First Int'l Workshop on Systems Management*, 1993, pp. 27–30.
12. N. Szabo, "Smart contracts: Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, Sep. 1997.
13. S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," http://nakamotoinstitute.org/bitcoin/, Visited 13 Nov 2017 2008.
14. M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. bft replication," in *Proc. Int'l Workshop on Open Problems in Network Security (iNetSec'15), LNCS*, 2015.
15. P. Bailis and A. Ghodsi, "Eventual consistency today: Limitations, extensions, and beyond," *ACM Queue*, vol. 11, no. 3, Mar. 2013.
16. C. Decker, J. Seidel, and R. Wattenhofer, "Bitcoin meets strong consistency," in *Proc. 17th Int'l Conf. on Distributed Computing and Networking (ICDCN'16)*, 2016.
17. J. Eberhardt and S. Tai, "On or off the blockchain? insights on off-chaining computation and data," in *(ESOCC'17)*, 2017.
18. G. Zyskind, O. Nathan, and A. S. Pentland, "Enigma: Decentralized computation platform with guaranteed privacy," arXiv.org, Tech. Rep. arXiv:1506.03471v1 [cs.CR], Jan. 2015.
19. D. Wörner and T. von Bomhard, "When your sensor earns money: Exchanging data for cash with bitcoin," in *Proc. ACM Int'l Joint Conf. on Pervasive and Ubiquitous Computing (UbiComp14)*, 2014.

20. bitinfocharts, "Cryptocurrency statistics," https://bitinfocharts.com, 2018.
21. J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," https://lightning.network/lightning-network-paper.pdf, Jan. 2016.
22. C. Molina-Jimenez, E. Solaiman, I. Sfyrakis, I. Ng, and J. Crowcroft, "On and off-blockchain enforcement of smart contracts," *arXiv preprint arXiv:1805.00626*, May 2018.
23. N. Minsky, "A model for the governance of federated healthcare information systems," in *IEEE Int'l Symposium on Policies for Distributed Systems and Networks (Policy'10)*, 2010.
24. G. Governatori, Z. Milosevic, and S. Sadiq, "Compliance checking between business processes and business contracts," in *Proc. 10th IEEE Int'l Enterprise Distributed Object Computing Conf. (EDOC'06)*. IEEE computer society, 2006, pp. 221–232.
25. O. Perrin and C. Godart, "An approach to implement contracts as trusted intermediaries," in *Proc. 1st IEEE Int'l Workshop on Electronic Contracting (WEC'04)*, 2004, pp. 71–78.
26. H. Ludwig and M. Stolze, "Simple obligation and right model (SORM)-for the runtime management of electronic service contracts," in *Proc. 2nd Int'l Workshop on Web Services, e–Business, and the Semantic Web(WES'03), LNCS vol. 3095*, 2003, pp. 62–76.
27. P. Gama, C. Ribeiro, and P. Ferreira, "Heimdhal: A history–based policy engine for grids," in *Proc. 6th IEEE Int'l Symp. on Cluster Computing and the Grid (CCGRID'06)*. IEEE CS, 2006, pp. 481–488.
28. C. Molina-Jimenez, S. Shrivastava, E. Solaiman, and J. Warne, "Contract representation for run-time monitoring and enforcement," in *IEEE International Conference on E-Commerce (CEC 2003)*, 2003.
29. E. Solaiman, C. Molina-Jimenez, and S. Shrivastava, "Model checking correctness properties of electronic contracts," in *Proc. Int'l Conf. on Service Oriented Computing (ICSOC'03)*. Springer, LNCS vol. 2910, 2003, pp. 303–318.
30. C. Molina-Jimenez, S. Shrivastava, E. Solaiman, and J. Warne, "Run-time monitoring and enforcement of electronic contracts," *Electronic Commerce Research and Applications*, vol. 3, no. 2, pp. 108–125, 2004.
31. S. Shrivastava, "An overview of the tapas architecture," http://tapas.sourceforge.net/deliverables/D5Extra.pdf, Jan 2005, supplement Delivery of the TAPAS (Trusted and QoS-Aware Provision of Application Services) IST Project No: IST-2001-34069.
32. N. Cook, P. Robinson, and S. Shrivastava, "Component middleware to support non-repudiable service interactions," in *Proc. IEEE Int. Conf. on Dependable Systems and Networks (DSN 2004)*, 2004.
33. R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution," *arXiv:1804.05141 [cs.CR]*, 2018.
34. F. Idelberger, G. Governatori, R. Riveret, and G. Sartor, "Evaluation of logic–based smart contracts for blockchain systems," in *Proc. 10th Int'l Symposium RuleML'16: Rule Technologies: Research, Tools, and Applications, LNCS, Vol 9718*, 2018, pp. 167183,.
35. C. Molina-Jimenez, I. Sfyrakis, E. Solaiman, I. Ng, M. W. Wong, A. Chun, and J. Crowcroft, "Implementation of smart contracts using hybrid architectures with on-and off-blockchain components," *arXiv:1808.00093 [cs.SE]*, 2018.
36. Ethereum, "Decentralized apps (dapps)," https://github.com/ethereum/wiki/wiki/Decentralized-apps-(dapps), 2018.
37. E. Solaiman, I. Sfyrakis, and C. Molina-Jimenez, "High level model checker based testing of electronic contracts," in *Cloud Computing and Services Science*. Springer-Verlag, LNCS Vol. 581, 2016.
38. I. Sergey, A. Kumar, and A. Hobor, "Scilla: a smart contract intermediate-level language: Automata for smart contract implementation and verification," https://arxiv.org/abs/1801.00687, Jan. 2018.