Computational Governance and Violable Contracts for Blockchain Applications

Munindar P. Singh and Amit K. Chopra

Abstract

We propose a sociotechnical, yet computational, approach to building decentralized applications that naturally accommodates and exploits blockchain technology. This approach avoids the shortcomings of smart contracts that arise from their regimented way of organizing computation, which limits their prospects for practical decentralized applications.

A centerpiece of our architecture is the notion of a declarative, violable contract in contradistinction to smart contracts. This new way of thinking enables flexible governance, by formalizing organizational structures; verification of correctness without obstructing autonomy; and a basis for trust.

Keywords: Smart contracts; Contracts; Sociotechnical systems; Blockchain

1 Introduction

Blockchain technology has brought newfound prominence to the challenges of building *decentralized systems*, which we understand quite literally as systems with no distinguished locus of control. As such, blockchain is a natural fit for building systems that support interactions among *autonomous* parties, each an independent locus of control. Unsurprisingly, blockchain promises support for multiparty interactions in domains such as government, health, manufacturing, and banking [1].

Blockchain applications, conceived to upend conventional business models, rely upon a *smart contract*—executable code placed and executed from a blockchain (Sidebar 1). But smart contracts suffer from major shortcomings that undermine their usefulness for decentralized applications. Specifically, smart contracts are antithetical to autonomy and compatible only with *endogenous* applications—those computed entirely within a blockchain. Thus, smart contracts are inadequate for real applications (consider healthcare, finance, or IoT), which typically involve external components.

Argument: Violability, Verifiability, Validation Decentralized applications presuppose modeling interactions between autonomous parties, which calls for a representation of contracts. A crucial property of any contract is *verifiability*: it should be possible to determine from a public record of events whether the contract was satisfied or violated. Verifiability lies at the heart of a *public semantics* [2].

Whereas smart contracts seek to prevent violation, we embrace *violability* and make verifiability explicit. *Verifiability* requires a formal representation of a contract to computationally evaluate a history of attestations. And, *validation*, ensuring that stakeholder requirements are correctly captured, presumes a high-level language that provides relevant abstractions.

Accordingly, we formulate a perspective on *sociotechnical systems (STSs)* whose salient features are *(1)* an autonomy-preserving representation for *violable* contracts; *(2)* guaranteed verifiability

through formal semantics interpreted over blockchain; *(3)* high-level representation to facilitate validation; and *(4)* an architecture of organizations that balances flexibility and rigor to engender trust.

Scope and contributions We focus on sociotechnical challenges, deemphasizing concerns such as confidentiality and performance, and contributing:

- An analysis of the shortcomings of smart contracts through the lens of decentralized applications.
- A formulation of research challenges to address those shortcomings from a sociotechnical perspective.
- A description of the key elements of a possible solution.

Sidebar 1: Blockchain and Smart Contracts, Conceptually

Notionally, a blockchain is an immutable distributed ledger, as epitomized by Bitcoin (https://bitcoin.org/bitcoin.pdf). Blockchain solves the longstanding distributed computing problem of achieving immutable agreement on the state of the system, despite failures and malice. Here, immutability relies upon consensus, which relies upon a majority of the computing power on the network remaining in the hands of benevolent (that is, protocol-following) parties. Specifically, blockchain determines a consensus order in which events have occurred.

The notion of a smart contract (https://en.wikipedia.org/wiki/Smart_contract) predates blockchain. A smart contract specifies contractual conditions programmatically, such that the contract would automatically execute when input data meets the stated conditions. A vending machine is characterized as a smart contract that takes in coins and outputs a product. Smart contracts could potentially be attached to any real-world object, e.g., a house for rent (https://slock.it).

In blockchain applications, a smart contract is digitally signed by its creator and placed on a blockchain. Since a smart contract is public, the parties wishing to exercise it can know in advance how it will function—provided they can understand the associated program. Hence, smart contracts can enable commerce in an open setting.

Bitcoin transactions are simple smart contracts—Bitcoin's limited language allows little more than verifying signatures. But subsequent approaches, including Ethereum, ambitiously support Turing-complete languages for smart contracts that initiate transactions based on observed events.

2 Sociotechnical Limitations of Smart Contracts

Let's consider the hazards of smart contracts. The Decentralized Autonomous Organization (DAO) fiasco (https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability/) is telling. DAO, a venture funding entity created as a smart contract on the Ethereum blockchain, was hacked to the tune of $50M by exploiting a flaw in the smart contract and the underlying Ethereum virtual machine. The specific flaw does not concern us since it is merely a symptom of a flawed architecture that confuses verifiability with inviolability.

Interestingly, the hack was remedied by causing a fork in the blockchain. Specifically, several Ethereum users colluded to extend a prior block as a way to exclude the undesirable transactions,

discarding legitimate ones as well. (Naturally, this effort produced two competing versions of Ethereum, though the details of their history don't concern us here.) Of course, a fork was possible only because a large fraction of the active participants agreed to it. A minority would not be able to take such remedies.

For something like DAO, it may be appropriate to discard several days of legitimate transactions to avert a loss of $50M. But what would the tradeoffs be in practice? Would it be fair to discard an hour's worth of real commerce at the national scale to save $50M? We suspect not. A patient attacker may succeed by causing only small amounts of harm at a time, for which detection and reversion are infeasible.

The success of the fork, however, *undermines the very point* that motivated blockchains, namely, their immutability. The episode reinforces the main claim of this paper: There is necessarily a social underpinning to any approach that has pretensions to decentralization. On permissioned blockchains such as Hyperledger (http://hyperledger.org), where membership is controlled, the risk is presumably better contained. However, errors in smart contracts are unavoidable and undesirable outcomes would be difficult to reverse.

In essence, our main choice is *(1)* whether to keep the social component ad hoc, hidden, and second class—as existing approaches do; or *(2)* to make the social component principled, explicit, and computational—as we propose doing in this article.

We now discuss three major shortcomings in the current conception of smart contracts and formulate questions that guide our investigation.

2.1 Lack of Control

The independence of participants with respect to their beliefs and actions is a crucial aspect of decentralization. Blockchain supports independence with regard to private beliefs since consensus applies only to shared events, such consensus being essential for achieving interoperation.

However, smart contracts fail independence for actions. They automate processing, removing control from the participants. A smart contract once launched cannot be overridden. Indeed, we cannot even contemplate overriding a smart contract because it executes automatically.

How can we reconcile blockchain with participant autonomy?

2.2 Lack of Understanding

Since the meaning of a smart contract is hidden in a procedure, even though it may be public, one cannot readily determine whether it meets stakeholder requirements, and how it may be exercised by a participant. Since blockchains are immutable, any mistake in capturing requirements cannot be corrected without violating immutability. Therefore, a powerful language for smart contracts placed on a blockchain poses a huge risk, as the DAO incident illustrates.

Instead, we need a language in which we can capture the essential stakeholder requirements directly. To enhance confidence in capturing requirements correctly, such a language would offer constructs close to the stakeholders' conception and would be limited in expressiveness.

How can we develop a contract language including an appropriate semantics?

2.3 Lack of Social Meaning

Any software application involves contact with the real world. In rare cases, the real world can be abstracted out. Bitcoin, being designed for cryptocurrency, is endogenous, meaning that bitcoins exist entirely within the blockchain, which can therefore ensure their integrity. Bitcoin is an atypical blockchain application since it excludes considerations other than of transactions involving bitcoins.

More commonly, applications such as healthcare and commerce are entwined with the real world, both social and technical. For example, in healthcare, surgical equipment may fail or a patient may deny having been adequately informed when giving consent. For physical or communication failures, the possible resolutions lie in the social sphere, as traditionally handled through contracts and laws.

The DAO hack demonstrated an integrity violation, indicating a platform failure. In a decentralized scenario, any response to an interoperation failure, including a platform failure, *must* be social. Indeed, the response to fork the Ethereum blockchain was social—it's just that it was an ad hoc and unverifiable response entirely outside the computational realm.

How can we enhance blockchain with abstractions to express and compute with social meaning?

3 Architecture: Compacts, Governance, Verification, Trust

The foregoing discussion shows that smart contracts are inadequate for describing interoperation between autonomous parties: they take over control of participant decision making, are opaque, and omit social meaning. We now describe our architecture that avoids these shortcomings and enables natural interactions between autonomous parties.

3.1 Declarative Violable Contracts

We introduce the term *compact* (https://www.ldoceonline.com/dictionary/compact) for our conception of contracts to avoid confusion with both smart contracts and traditional contracts.

In contrast to a smart contract, a compact is not a program executed by the blockchain but a *specification of correct behavior*. In contrast to a traditional contract, a compact is a computational artifact: its formal semantics determines which blockchain instances satisfy and which violate the compact. A compact would be stored on the blockchain and be unambiguously computed based on its semantics.

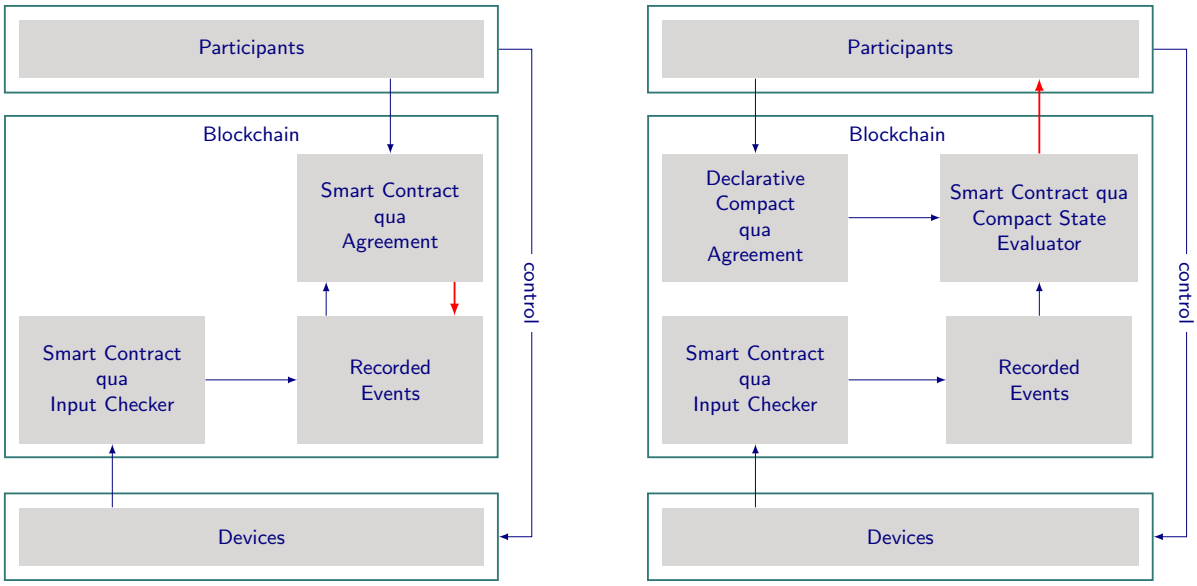
Figure 1 illustrates how compacts differ from smart contracts. In both settings, principals (social entities) own and control devices (technical entities), such as computers, sensors, and vehicles. Importantly, the autonomy rests with social entities who control the technical entities.

A device may originate an event or relay an event from another source, such as a human. The blockchain records events received from devices provided they pass any input checks. Smart contracts provide two functions in both architectures and, in both, the input checker is a smart contract.

In the traditional architecture, in Figure 1a, the principals additionally specify their business agreements as smart contracts that carry out actions and record events on the blockchain. Thus, a smart contract once launched may perform immutable (modulo rollback and forking, as in the DAO incident) changes to the blockchain.

In our proposed architecture, in Figure 1b, the principals specify the compacts corresponding to their business agreements. Given the recorded events, the evaluator—a smart contract by fact of being a program on the blockchain—determines whether a compact is satisfied, violated, expired, or neither. It informs the principals about states of relevant compacts, but does *not* insert events into the blockchain. That is, of the functions of smart contracts in Figure 1a, Figure 1b retains being able to check (filter) incoming events and query the blockchain but not to make changes to the blockchain.

In general, our sociotechnical architecture splits the necessary functions between the social and technical parts (and models the social part computationally) whereas the traditional architecture hides the social part and places all functions in the technical part. Through our models of compacts, organizations, and trust, the rest of this paper demonstrates how the social and technical parts



(a) Smart contracts not only check received events but can insert additional events into the blockchain.

(b) Compacts provide a declarative standard of correctness but do not insert events into the blockchain.

Figure 1: Comparing compacts and smart contracts in a blockchain architecture.

coexist. For expository convenience, we place input checking as a technical function; in the limit, we could potentially dispense with input checking and handle all discrepancies at the social level though it would not be an effective approach for many practical applications.

A compact helps balance autonomy and correctness. A party to a compact, in exercising its autonomy, may violate the compact. For example, a compact in healthcare may specify that a hospital prohibits a nurse from sharing a patient's data without the patient's consent. Yet, a nurse Bob may share patient Charlie's data with cardiologist Alice without Charlie's consent. From the semantics, given recorded events on the blockchain, we can compute whether the compact was satisfied or violated. Crucially, violation doesn't entail malfeasance. It could be that Charlie had a medical emergency and was in no condition to give consent. Bob could be rewarded for saving Charlie's life for his workaround [3].

Consider another example: a compact for renting apartments to tourists. Such a compact may stipulate conditions such as that registered guests may not smoke in the apartment; not invite others except children under twelve years old to stay overnight; and not leave the windows open during the day. Such prohibitions are impossible to impose through the blockchain since they concern exogenous events and would be impractical or risky to enforce physically. Guests may violate the stipulations in the compact. The rental agency may install monitors (e.g., smoke detectors, face recognizers, and window sensors) that enable detecting violations of the compact. Again, from the semantics and recorded events, we can compute whether the compact was satisfied or violated. Again, violation doesn't entail malfeasance. First, sensors aren't perfect and a smoke detector may falsely report smoking, e.g., because of a deep fryer. Second, a compact may be overridden by other compacts (don't leave a child alone) or a principal may discover the circumstances are such that it is sensible to violate a compact. For example, if the adult guests are taken to a hospital, they may hire a baby sitter to stay overnight with their young children even though it is prohibited by the compact.

The foregoing examples highlight the importance of compacts in detecting and resolving con-

flicting requirements [4].

3.2 Specifying Compacts via Norms

To recover understanding, control, and make the social meaning explicit, we need a declarative representation for compacts that captures the essence of traditional contracts. A compact would explicitly state what each concerned party may expect from another. To this end, the formal notion of *norms*, which resembles but is not identical to "social norms" in the vernacular, yields promising constructs. As motivated by Georg von Wright, who invented modern deontic logic in the 1950s, this notion of norm carries regulatory force [5].

Therefore, we propose to represent each compact as a set of norms. The specific norms we adopt are *commitment*, *authorization*, *prohibition*, and *power*. The following are key features of norms.

- Each norm in our representation is directed from its Accountable Party to its Party with Standing [6]. Thus, a norm always makes accountability clear.
- Each norm arises in the context of an organization. Thus, a norm makes its scope and adjudicating jurisdiction clear.
- Each norm is conditional, and states logical conditions under which it goes in force (antecedent) and under which it completes (consequent). The antecedent and consequent are definitively evaluated on a ledger, thereby ensuring clarity on what state each norm instance is in.

Let's introduce our specification language, based on the Custard language [7], via example. Let's begin with a fairly routine business agreement, which may be described by a compact comprising the following commitment. In the commitment, keywords are in sans serif. Words beginning with an uppercase letter are names of event schemas unless otherwise specified. Words beginning with a lowercase letter are attributes of the events.

```
compact Market
role Seller Buyer Marketplace

commitment DiscountQuote from Seller to Buyer within Marketplace
  create Quote
  detach (Order and Payment) deadline Quote + 10m
    where paymentAmount >= 0.90 * quotedPrice * quantity
  discharge Shipment deadline Payment + 5d
```

This listing describes a compact, Market, consisting of one commitment schema labeled DiscountQuote, which is directed from a role Seller to a role Buyer. At runtime, these roles are played by specific principals, e.g., individuals Meryl and Custer.

- An instance of DiscountQuote is created when an instance of the *Quote* event occurs—*Quote* being the event expression given under create. The attributes of *Quote*, such as quoteID, item, and quotedPrice, are the information relevant to the creation of this commitment. As Seller, Meryl alone can commit herself.
- A (created) instance of DiscountQuote is detached when instances of *Order* and *Payment* for a matching quoteID occur within 10 minutes of the matching *Quote*, and paymentAmount is at least 90% of the cost of the *Order* items (quotedPrice × quantity). Here, Custer would bring about those events although in general a commitment could be detached through anyone's actions.

- An instance of DiscountQuote expires if a matching *Order* and *Payment* do not occur within 10 minutes of the matching *Quote*.
- If *Shipment* for the matching quoteID occurs (and, if there is a matching instance of *Payment*, occurs within five days of *Payment*), the commitment is discharged. Presumably, Meryl or one of her business partners would bring about this event.
- If the commitment is detached, but *Shipment* for the matching quoteID does not occur within five days of *Payment*, then the commitment is violated. Now Custer can hold Meryl to account.

We illustrate the above healthcare example. For brevity, we focus on a prohibition norm and assume the relevant events: Employment, when a Nurse becomes employed by a Hospital; DataAccess, when a Nurse accesses a Patient's data; and CopyData, when a Nurse shares a Patient's data with someone. Attributes of these events express relevant information, including the Patient's identity.

The compact specifies a prohibition on a Nurse by a Hospital that is created when the Nurse is employed. When the Nurse accesses a Patient's data, the Nurse may not copy that data to anyone outside of the Patient's care team.

```
compact PatientData
role Patient Hospital Nurse MedicalSystem CareTeam

 prohibition NoSharing on Nurse by Hospital within MedicalSystem
  create Employment       /* in Hospital*/
  detach DataAccess       /* about Patient*/
  violate CopyData        /*to receiver*/
   where receiver not in CareTeam of Patient /*receiver is an outsider*/
```

3.3 Computing the Norm Lifecycle

The above declarative specification of compacts yields significant benefits over smart contracts. First, the language of norms is geared toward expressing agreements between autonomous principals: norms can be reliably identified from real-life natural language contracts [8]. Two, the language is amenable to formal reasoning since it is simpler than a traditional programming language. Three, specifications in the language can be automatically evaluated, meaning that the state of any norm can be unambiguously determined from the norm's expression and the events recorded in any snapshot of the blockchain.

Specifically, a blockchain is naturally modeled as a sequence of events with timestamps. From these events, we can determine what norm instances have been created, and which of them have transitioned to other relevant states. For example, we might observe that Meryl has produced instances of *Quote* for ten prospective buyers, of whom Custer alone has responded with matching instances of *Order* and *Payment*. From the matching *Shipment* event, we can conclude that Meryl discharged her commitment to Custer.

Alternatively, if Meryl failed to bring about a matching *Shipment* event, we would conclude that Meryl violated her commitment to Custer. Importantly, we can compute abstract events, such as when a norm instance transitions in its lifecycle. For example, the violation of a commitment is itself an event that we can effectively compute. That event could be referenced from other norms—essential to achieving governance, as discussed next.

3.4 Organizations and Governance

Consensus on what has transpired can support decentralized applications by averting disputes as to the public facts. But, as envisioned here, the principals may nevertheless violate applicable compacts.

Decentralized applications cannot avoid governance: the choice is whether to leave governance ad hoc and manual or to make governance formal and computational, as we envision. In our conception, every decentralized application is associated with an organization, which serves as the context of its defining compact. Such organizations are seen on today's blockchains, such as channels on Hyperledger Fabric. However, current practice doesn't model the organization itself. The ill-fated DAO was arguably modeled procedurally as a smart contract but that is not satisfactory since, even if it were correct, its behavior would not have been comprehensible or modifiable.

Today's approaches lack a computational model for such organizations. Consequently, there is no precise characterization of what an organization can expect from its members and vice versa. As a result, governance in blockchain applications remains ad hoc.

To address this limitation, we propose a three-pronged approach. First, we model an organization as a principal on par with any other, such as an individual. An organization may feature as a subject or object of another norm.

Second, an organization provides an *organizational context* for each norm arising, as described in [9]. In the listing above, the role Marketplace provides the context for Seller and Buyer's dealings. The Marketplace role would be adopted by a concrete organization, such as Raleigh's Artsplosure (an arts fair), within whose scope Meryl and Custer would interact if they joined Artsplosure. The context can embody jurisdictional weight and can serve as an adjudicating authority for disputes. The context can thus help mitigate violations of norms in a compact [10].

The Marketplace serves as the context for the DiscountQuote commitment and the MedicalSystem as the context for the NoSharing prohibition. The organizational context is captured as a role in its own right. Let us extend the above examples to illustrate how compacts can handle violations. Marketplace commits to Buyer that if a Seller violates the DiscountQuote commitment, Marketplace would step in and provide a refund within two days.

```
commitment Compensation Marketplace to Buyer within Marketplace
create Quote                          /* when Seller creates a Quote */
detach violated(DiscountQuote)
discharge Refund deadline violated(DiscountQuote) + 2d
 where refundAmount = paymentAmount
```

Similarly, Hospital, as the context of the NoSharing prohibition norm, commits to Patient to investigate any violations of the NoSharing prohibition within 30 days.

```
commitment SanctionC from Hospital to Patient within MedicalSystem
create Enroll                  /* when Patient enrolls */
detach violated(NoSharing) /* the Patient ID match is implicit */
discharge Investigation deadline violated(NoSharing) + 30d
```

Now when Bob reveals Charlie's data without Charlie's consent, the hospital's commitment to Charlie is activated. The hospital can satisfy its commitment by conducting its investigation, upon which it may either exonerate and reward Bob or penalize him.

Third, the organization is specified through a compact between itself and its members. This compact specifies precisely what expectations an organization and its members may have of each other. Membership in the organization provides identity for all purposes within that organization. Enrollment as member *may* rely upon another organization that this organization is part of, where the second organization provides its identity, and so on. The nesting would ordinarily terminate

either at a self-contained organization (as in Bitcoin) or at the real society (as in the banking industry, where regulations require a national ID for each depositor). Certain organizations, such as for social services to drug users, may be self-contained to protect the anonymity of the people they help.

3.5 Programming and Verifying Interactions

Achieving coordination is nontrivial in decentralized applications. Existing approaches hardcode coordination in software implementations. Doing so reduces flexibility in interoperation and hides essential details, thereby preventing composing compacts. Blockchains, e.g., Hyperledger, provide coordination abstractions such as a *channel*—a subnet on which only participants can access information. A channel supports confidentiality and helps decouple participants by hiding irrelevant information. To enable interoperation, we must formalize how an interaction proceeds, not just who participates or what data they exchange. Thus, we face challenges of how to specify a channel and to produce software to interact through a channel.

An effective solution would specify coordination declaratively in conjunction with compacts. Doing so requires not just formal semantics for data [11] but also models of causality and integrity constraints on interactions underlying the data [12].

In essence, we would formally specify an interaction *protocol* for each compact that would ensure the compact can be flexibly enacted, meaning that the protocol does not foreclose any enactment that remains acceptable with respect to the compact. Specifically, the protocol includes a way for each lifecycle state of each norm (including states of satisfaction and violation) in the compact to be realized. To capture the intuition that a decentralized application is specified via a compact, we would need to generate protocols automatically from a compact such that each involves only the relevant principals. The interactions in the protocol would naturally be endowed with a *public semantics* [2], a major benefit of a shared ledger.

The input checker component, realized as a smart contract in Figures 1a and 1b, helps ensure integrity of the information in the blockchain. Thanks to our approach being based on compacts, we can produce the checker's specification from a compact, and the specification can capture, for each principal, the legal actions with respect to the compacts in which that principal participates.

3.6 Meaningful Trust and Reputation

The autonomy of principals and embedding in the real world suggest that principals would need to trust one another to interoperate. The possibility of violation of a compact creates a *vulnerability*, a hallmark of trust [13]. Blockchain obviates the need for trust only to the extent that the governance structures provide assurance against malfeasance by another—and the structures themselves are trusted.

The Compensation example above illustrates how to achieve coherent interactions without a central authority [6, 14, 15]. Governance is a prerequisite for accountability and trust, which are means with which to balance autonomy and correctness.

Blockchain can serve as a platform for promoting meaningful trust. First, quite naturally, the states of relevant compacts provide an opportunity to make evidential trust judgments. Violation and satisfaction of a norm would mean a lowering and raising, respectively, of trust in the concerned party with respect to similar norms. Second, explicit governance engenders trust by assuring principals that malefactors would be sanctioned. A party may violate a compact by failing to satisfy its conditions, but if it does so its violation would be determinable from the blockchain. The

aggrieved party (http://thelawdictionary.org/aggrieved-party/) may file a complaint, also recorded in the blockchain, thereby triggering a governance compact.

Third, governance provides a basis for capturing the trust assumptions by formalizing what counts as evidence for what norm. Consensus on blockchain concerns the events observed. But armed with a governance structure, we can encapsulate norm-relevant evidence within an event to reflect the application meaning. For example, a norm may rely upon a patient having a benign tumor. But, in medical practice (http://aspe.hhs.gov/sp/reports/2010/PathRad/index.shtml), whether a tumor is benign is a fact that is established by the tumor board of the hospital. That is, the tumor board's assertion counts as the tumor being benign.

4 Prototype over R3 Corda

Our compacts-based approach is readily realized on existing blockchains. To illustrate our approach, we implemented a proof of concept prototype on R3 Corda (https://www.r3.com/). The relevant Corda programming abstractions are these: Corda is a network of *nodes*, each of which hosts a relational database. Each *party* is a business entity and runs a node. A *workflow* is a program over database *transactions* that specifies the parties to whom the transaction is visible. By invoking a workflow, a party invokes the transactions in the workflow, which causes updates to its own database and the databases of the other parties to whom the transaction is visible. An *RPCclient* is a means by which a party may invoke a workflow programmatically.

Our prototype maps the principal and event constructs to Corda's party and transaction, respectively. It specifies a workflow for each event and encodes agents as RPCclients that invoke the workflows to insert events into the databases of the appropriate parties. The prototype adapts our norms compiler [7] to generate queries for the lifecycle states of each norm. These queries execute on a node database. Additional details are in the supplementary material.

5 Discussion

The emergence of blockchain as a platform for decentralized applications exposes new usage scenarios. These scenarios bring forth sociotechnical considerations into computing—specifically, in terms of expectations regarding governance (organizations, norms, privacy) and trust.

Table 1 highlights how our architecture of compacts contrasts with existing approaches. In the compacts approach, the blockchain declaratively represents contractual relationships; maintains relevant events; enables a principal to violate a compact if the principal so desires; computes whether the compact is satisfied, violated, expired, or otherwise pending; thereby activating applicable governance compacts and providing a basis for trust.

In this manner, we envision computational representation and reasoning about sociotechnical considerations. Specifically, we advocate developing approaches for programming interactions using blockchain that build on and support effective governance and trust. In this way, we differ from the notion of Ricardian contracts (https://iang.org/papers/ricardian_contract.html), which associates a textual description with a computational description, thereby creating two competing standards of correctness: one that is applied computationally and one that is understood by people. In our approach, there is only one standard—it is high-level (so understandable by people) and computational (so executable by machine).

The compacts-based architecture yields valuable research opportunities concerning how principals *(1)* preserve autonomy in being able to violate a compact and verify each other's compliance;

Table 1: Contrasting compacts with traditional and smart contracts.

	Traditional	**Smart**	**Compacts**
Specification	Text	Procedure	Formal, declarative
Automation	None	Full	Compliance checking
Principals' Control	Complete	None	Complete
Venue	External	Within blockchain	Recorded on blockchain
Trust Model	Hidden	Hardcoded	Explicit
Social Meaning	Informal	None	Formal
Standard of Correctness	Informal legal	Whatever executes	Formal legal
Scope	Open but ad hoc	Closed	Sociotechnical

(2) deal with events in the real business or social worlds, external to the blockchain; *(3)* maximize flexibility in having their interactions minimally constrained to interoperate successfully; and *(4)* most importantly, build and realize governance structures to deal with autonomy and exceptions.

Acknowledgments

Singh was supported by an IBM Faculty Award and Chopra by EPSRC grant EP/N027965/1 (*Turtles*). We thank Alessandra Scafuro and Samuel Christie for helpful discussions. Thanks also to the anonymous reviewers for their helpful comments.

References

[1] Daniele Magazzeni, Peter McBurney, William Nash. Validation and verification of smart contracts. *IEEE Computer*, 50(9):50–57, 2017.

[2] Munindar Singh. Agent communication languages. *IEEE Computer*, 31(12):40–47, 1998.

[3] Ross Koppel, Sean Smith, Jim Blythe, Vijay Kothari. Workarounds to computer access in healthcare organizations. In Karen Courtney, Alex Kuo, Omid Shabestari, editors, *Driving Quality in Informatics*, pages 215–220. IOS Press, 2015.

[4] Jéssica dos Santos, Jean de Oliveira-Zahn, Eduardo Silvestre, Viviane da Silva, Wamberto Vasconcelos. Detection and resolution of normative conflicts in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 31(6):1236–1282, 2017.

[5] Georg Von Wright. Deontic logic. *Ratio Juris*, 12(1):26–38, 1999.

[6] Munindar Singh. Norms as a basis for governing sociotechnical systems. *ACM Trans. Intelligent Systems Technology*, 5(1):21:1–21:23, 2013.

[7] Amit Chopra, Munindar Singh. Custard: Computing norm states over information stores. *Proc. Autonomous Agents and Multiagent Systems*, pages 1096–1105, 2016.

[8] Xibin Gao, Munindar Singh. Extracting normative relationships from business contracts. *Proc. Autonomous Agents and MultiAgent Systems*, pages 101–108, 2014.

[9] Munindar Singh. An ontology for commitments in multiagent systems. *Artificial Intelligence and Law*, 7(1):97–113, 1999.

[10] Munindar Singh, Amit Chopra, Nirmit Desai. Commitment-based service-oriented architecture. *IEEE Computer*, 42(11):72–79, 2009.

[11] Allan Third, John Domingue. Linked data indexing of distributed ledgers. *Proc. World Wide Web Companion*, pages 1431–1436, 2017.

[12] Munindar Singh. Semantics and verification of information-based protocols. *Proc. Autonomous Agents and MultiAgent Systems*, pages 1149–1156, 2012.

[13] Cristiano Castelfranchi, Rino Falcone. *Trust Theory.* Wiley, 2010.

[14] Jeremy Pitt, Alexander Artikis. The open agent society. *Artificial Intelligence and Law*, 23(3):241–270, 2015.

[15] Christopher Frantz, Martin Purvis, Mariusz Nowostawski, Bastin Savarimuthu. nADICO: A nested grammar of institutions. *Proc. Principles and Practice of Multi-Agent Systems*, LNCS 8291, pages 429–436, 2013.

Author Bios

Munindar P. Singh is a Professor in Computer Science and a co-director of the Science of Security Lablet at NC State University. His research interests include the engineering and governance of sociotechnical systems, including the applications of blockchain in automating financial transactions. Singh is an IEEE Fellow, a AAAI fellow, and a former Editor-in-Chief of *IEEE Internet Computing* and *ACM Transactions on Internet Technology*. Contact him at singh@ncsu.edu.

Amit K. Chopra is a senior lecturer in the School of Computing and Communications at Lancaster University in the UK. Chopra's interests span sociotechnical systems, multiagent systems, and decentralized systems. Contact him at amit.chopra@lancaster.ac.uk.