September 2020

Digital Enforceable Contracts (DEC): Making Smart Contracts Smarter

Lu-Chi LIU [a,1], Giovanni SILENO [a] and Tom VAN ENGERS [b,a]
[a] *Informatics Institute, University of Amsterdam, Amsterdam, Netherlands*
[b] *Leibniz Institute, University of Amsterdam/TNO, Amsterdam, Netherlands*

Abstract. The combination of smart contracts with blockchain technology enables the authentication of the contract and limits the risks of non-compliance. In principle, smart contracts can be processed more efficiently compared to traditional paper-based contracts. However, current smart contracts have very limited capabilities with respect to normative representations, making them too distant from actual contracts. In order to reduce this gap, the paper presents an architectural analysis to see the role of computational artifacts in terms of various ex-ante and ex-post enforcement mechanisms. The proposed framework is assessed using scenarios concerning data-sharing operations bound by legal requirements from the General Data Protection Regulation (GDPR) and data-sharing agreements.

Keywords. Smart contracts, Norm representation, Normative reasoning, Automated enforcement, Data sharing infrastructures, GDPR

1. Introduction

Smart contracts were originally motivated by the wider purpose of facilitating the performance and enforcement of traditional paper-based contracts [1], but today there is no reference in smart contracts to normative constructs as those to be found in legally binding contracts. Several studies have shown that it is possible to perform reasoning tasks on blockchain via smart contracts, typically by querying an off-chain, trusted oracle [2] (e.g. a reasoner module [3]); this integration enables more (normative) expressiveness. Enforcement can also be achieved by means of dedicated social infrastructures. For instance, [4] proposes a model of incentives to enable enforcement for off-chain activities. On similar lines, in production settings, consider e.g. Kleros [5], a blockchain-based decentralized application was developed for multipurpose dispute resolution; smart contracts can assign Kleros as their arbitrator infrastructure in case a dispute occurs (see also Codelegit, Juris, Oath, etc.). Yet, these solutions are heterogeneous in methods and cover different aspects of compliance. The core of our contribution is to frame the problem in terms of possible types of enforcement mechanisms, to provide a modular architecture that covers the distinguishable application types. Rather than focusing on technology-dependent functions of smart contracts, this paper advances the concept of *digital enforceable contract* (DEC), with the purpose of highlighting the higher-level functions that any sound computational normative artefacts are expected to provide.

[1] Corresponding Author: l.liu@uva.nl. This research is funded by the Dutch Organisation for Scientific Research (NWO) under contracts 628.009.014 (SSPDDP project) and 628.001.001 (DL4LD project).

2. Implementing Enforcement Mechanisms

Types of enforcement Enforcement mechanisms can be distinguished in two main approaches, depending on *when* they play a role with respect to violations: *ex-ante* (before the facts) and *ex-post* (after the facts). In computational domains, most of the attention is put on the first approach; in socio-legal settings the term typically refers to the second approach, plausibly because legal activity is usually triggered by the occurrence of violations. We identified the following patterns. To avoid violations to occur (*ex-ante enforcement*) it is necessary to check: (a) whether a program/an action is permitted before executing; (b) whether any positive duties holds. As to identify that a violation has occurred (*ex-post enforcement*), we need to check: (c) whether a prohibited action has been performed; (d) whether a positive duty has not been fulfilled. The last two conditions requires active monitoring by an 'enforcement agent', i.e. an agent that has an *institutional power* to force that the duty is fulfilled, or that some other remedy is provided, and/or to 'punish' the agent that hasn't fulfilled the duty.[2] Additional design dimensions can also be taken into account. First, all enforcement mechanisms rely on some conditions that need to be evaluated (about occurrence of events, or properties of agents, etc.). This evaluation can be *lazy* (computed only at the moment of need), or *eager* (as soon as relevant conditions become true). Second, regulation can be *internal* (the agent on itself) or *external* (by a enforcer agent). Third, the monitoring task can be *internalized*, as when they are set up by claimants of duties, or *externalized*, e.g. by some infrastructural component, or by witnesses.

A modular architecture for enforcement To deal with the richness observed above, we propose a architectural model consisting of a number of modules associated to dedicated control and enforcement mechanisms that can be imported at need. This minimal set of modules has been selected as capturing and providing the functionality necessary to run all enforcement constructs.

We assume a distributed computation setting, representative of e.g. data-sharing infrastructures. We consider as minimal unity of agency a computational **actor**, characterized with a name/id. In a data-sharing infrastructure one might expect actors running applications for *users* of the infrastructure, as well as actors running applications for the *owners/maintainers* of the infrastructure, as e.g. for enforcement purposes. Actors can be then decomposed to a number of components, having unique functions. A **program** is a list of instructions which can be regarded as a plan to achieve a given design goal associated to that actor. Actors can have more than one programs (plans) to opt from depending on the situation. An **executor** provides the internal control of the actor. It follows the execution of the currently selected program or modifies the control flow if needed. A **message queue** is the communication channel for actors to interact with each other. It delivers and receives messages to/from other actors. A **monitor manager** creates and destroys monitors which hook to certain events or facts. For example a user can set up a monitor to observe whether its action has failed or not, while an enforcer can set up monitors for violations. A **regulator** is the module dedicated to normative reasoning. It is initiated with specifications of norms and should be fed up with factual data. It

[2] Considering power merely as a conditional obligation (e.g. if this action is performed, then this duty comes to hold), one cannot model the fact that powers themselves can be created and destroyed, depending on the dispositions set by the contract. The ex-ante/ex-post distinction needs thus to be inflected to the case of power.

provides regulatory information, for example whether a certain instruction or program is permitted to run, and/or whether the instruction will lead to any positive duties.

Technically the regulator can be realized as an external component to the actors as well, so that a group of actors can share the same normative reasoner. It can be regarded as a "legal" consultant who provides conclusions from the associated normative specifications when queried by the executors with some input information. The interactions of a shared regulator are not functionally dissimilar from that of any other actor, therefore this module follows the same communication channels used by other actors. With respect to content, a few query templates can be identified for the communication between executor and regulator: e.g. (a) What position do self/other actor have now (with respect to a certain action)? (b) If self/other actor performs a certain action, what kind of position self/other will take? (c) Given a source self/other actor's position, which actions can self/others perform in order to reach a certain target position?

3. Proof of Concept

To provide an example of application, we assessed the proposed architecture on a data-sharing scenario in a context relevant to the GDPR. According to the GDPR, the data-controller needs to have consent from the data-subject to process his/her data. Once given, the data-subject can at any moment modify or revoke his/her consent. We modeled this normative content into a logic-based representation[3] and set up a server interfacing with a suitable reasoner. This server acts as an externalized regulator, and is implemented as an actor itself, receiving normative requests from other actors and answering them.

We considered then three agents/actors: (1) a data-controller "Bank", (2) a data-subject "John", and (3) a data-processor "Adcom". For the three actors, we created possible actions which could be performed to interact with each other, for example "give consent", "share data" and "send advertisement". Finally, for each type of enforcement we set up a *simulation* to test whether the proposed solution functions as expected.[4] In the following paragraphs we show how our proposed architecture and the reasoning mechanism can be used for ex-ante enforcement as well as for ex-post enforcement.

Ex-ante enforcement for permission checking The Bank attempts to use John's data for analysis. By sending a query regarding the permission of such action to the regulator, the regulator will inform the executor of Bank that, for being allowed, it must obtain a consent from John. The Bank will then send this request to John, setting up an adequate monitor for the reply. At the reception of the message, the executor of John will select the program giving consent and execute it. A new message is created and sent to the requesting Bank. This message is captured by the monitor of Bank and eventually delivered to its executor, which is now aware of the consent and can start using John's data.

Ex-ante enforcement of positive duties Continuing the previous scenario, now John (the user) changes his mind. He wants to modify his consent by asking the Bank to replace the old purpose "data analysis" to "marketing". By means of its regulator, Bank is aware

[3]More concretely, the models we used were written in eFLINT [6], a language for specifying policies based on normative frames. Note however that any other choices would have been equally good in functional terms.

[4]The code used for the proof of concept can be found on `https://gitlab.com/evelynliu324/digital-enforceable-contract`.

that the modification consent request will lead to a duty and can thus set up a monitor for it. Receiving the request message from John triggers the monitor, creating a notification about taking action.

Ex-post enforcement of violations of duty Adcom acts as a service provider for Bank to place advertisements. Since John now consents to have his data used for marketing purpose, Bank is permitted to share his data to Adcom. However, after receiving too many advertisements, John revokes his consent and requsets to remove his data. According to the GDPR, Bank, as data-controller, has the duty to fulfill these requests. In the model we set up a made-up norm that if the data-controller did not respond to the request within two weeks, the duty would be regarded as being violated. On this basis, John sets up a monitor with a timeout mechanism to check for violation. When the duty is due and not fulfilled, the monitor will send a message to the executor, notifying a violation of duty.

Ex-post enforcement of violations of prohibitions Even if removal and revocation have been confirmed, John sets up monitors for Bank and Adcom for known illicit behaviors. Suppose that Bank has performed the duty to revoke John's consent but Adcom keeps processing John's data sending promotions to John. As a result, the monitor notifies the reception of advertisements from Adcom, concluding that this is a violation of prohibition. The executor of John consults the regulator for deciding how to act upon it.

4. Conclusions

The limited norm representation capability makes smart contracts unable to work with norms similarly to traditional paper-based contracts. To address this limit, this paper started investigating architectural possibilities for *digital enforceable contracts*, with reasoning capacities enabling ex-ante and ex-post enforcement through an integrated normative reasoner and possibly a dedicated social infrastructure. The usability and effectivity of the proposed framework have been assessed by a practical scenario case involving data sharing operations subjected to the GDPR, showing that the overall architecture is sound and can support automated enforcement. The example was sufficient for our purposes, but we do not claim that it covers all perspectives and complexity of real-world contracts. In typical data-sharing environments, operations are not only subjected to the GDPR, but also many other regulations. Second, a proper ex-post enforcement requires an adequate design, reflecting the essentials of social infrastructures and the capacity to automatically find solutions, remedies or repairs based on diagnostic modules.

References

[1] Nick Szabo. Smart contracts: building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought,(16)*, 18:2, 1996.

[2] Luc Desrosiers and Ricardo Olivieri. Extend your blockchain smart contracts with off-chain logic, 2018.

[3] Michele Ruta, Floriano Scioscia, Saverio Ieva, Giovanna Capurso, Agnese Pinto, and Eugenio Di Sciascio. A blockchain infrastructure for the semantic web of things. In *SEBD*, 2018.

[4] Huan Zhou, Xue Ouyang, Jinshu Su, Cees de Laat, and Zhiming Zhao. Enforcing trustworthy cloud sla with witnesses: A game theory–based model using smart contracts. *Concurrency and Computation: Practice and Experience*, page e5511, 2019.

[5] Clément Lesaege, Federico Ast, and William George. Kleros: Short paper v1.0.7. 2019. `https://kleros.io/assets/whitepaper.pdf`.

[6] L. Thomas van Binsbergen, Lu-Chi Liu, Robert van Doesburg, and Tom van Engers. eflint: a domain-specific language for executable norm specifications. In *Proceedings of GPCE '20. ACM*, 2020.