How smart contracts can implement **“report once”**

Marc Sel, Henning Diedrich, Sander Demeester, Harald Stieber

marc@marcsel.eu(*), *h@claryon.com*, sander.demeester@pwc.com, *hstieber@wcfia.harvard.edu*
(*) corresponding author

Abstract

This paper explains the main features of and motivation for the “report once” demonstrator[1] shown at the 2017 Data For Policy conference. It shows how Ethereum[2] smart contracts, based on the semantics and algorithmic representations defined in ACTUS[3] can implement “digital doppelgängers” of financial contracts.

The implementation makes use of a private[4] Ethereum blockchain, with smart contracts written in Solidity.[5] The limitations of using ACTUS in a semi real-time scenario are explored, as well as how to overcome these limitations.

The major innovation, visualized by the demonstrator, is that compliance reports can be generated in semi real-time, using the information present in the “digital doppelgängers”, residing in the blockchain.

The demonstrator supports various use cases, illustrated through the narration of stories. In these stories, Alice, Bob and Eve are contracting parties, Romeo acts as regulator, and the narrator tells the stories. The stories cover trading a Bond, trading an Interest Rate Swap (IRS), the defaulting of a party (i.e. a payment stop), and various types of regulatory reports.

Keywords: *report once; smart contracts; blockchain; Ethereum; financial contracts; algorithmic standards; RegTech; digital doppelgänger; supervisory reporting*

1 Introduction

In this paper we discuss our implementation, in blockchain smart contracts, of the digital equivalent of a set of financial documents that represent a financial instrument. This representation is called a “digital doppelgänger”. While it does not have legal value and cannot be used for the transfer of (monetary) value, its objective is to capture and represent financial contract states at any given time and thereby allow for highly granular and semi-real time modelling of a financial contract's life-cycle, mirroring specific, real contracts.

Our approach builds on Kavassalis et al. (2016) who outline a vision of autonomous logical containers that capture real world financial contracts (e.g., loans, bonds, derivatives) [1]. We focus on efficiency gains of exploiting the inherent logic of financial instruments [2] which in combination with external events (hard-coded in this version of the demonstrator) allow for a comprehensive digital representation of the contract state [3] that can be reported at virtually zero marginal cost. Using this approach simplifies the compliance landscape that financial institutions are subjected to. Specifically, expressing the behaviour of real world contracts through algorithms obsoletes a class of complex challenges related to structuring reporting data that at this point deny high fidelity reporting aggregates for the regulator. It also opens new paths to performing more relevant *stress tests* on financial institutions.

2 Objective

The objective of the demonstrator is to reflect the state of traded “real world” financial contracts over their life cycle and to allow semi real-time reporting by implementing “digital doppelgängers” that reflect the status of such instruments at any point during this cycle. In the search for a RegTech solution where a high degree of data security is

[1] A first version of this demonstrator was built by PwC Enterprise Advisory and Claryon for the European Commission, DG FISMA (contract ABCIII-000428 under framework contract DI/07171) in the context of a multi-annual ISA[2] action on modern Financial Data Standards to enhance interoperability between public authorities as well as between private firms and public authorities that exchange data for the supervision of financial risk; details of this project can be found here: https://ec.europa.eu/isa2/home_en

[2] Ethereum is the pioneering blockchain focused on smart contracts, agreements that are guaranteed to execute and are expressed in computer code.

[3] “The ACTUS Algorithmic Standard unambiguously defines the logic embedded in legal financial agreements according to which contract terms are mapped to a stream of cash flows, or business events, respectively.” – http://actusfrf.org/index.php/algorithmic-standard/

[4] A private blockchain is technically a clone of the public mainnet of a blockchain that holds the cryptocurrency, in Ethereum’s case, Ether. While the mainnet is always-on and supported by thousands of nodes, a private network can consist of few or only one node, can be started and stopped at any time and usually does not carry an accepted cryptocurrency.

[5] Solidity is the most popular high-level language that can be compiled to the bytecode that express smart contracts on an Ethereum blockchain.

achieved [4] at low set-up and operating costs, the demonstrator thus serves as a tool to better understand how to meet the regulatory reporting requirements[6] in an effective and efficient way. In the context of the project for the European Commission, it allowed us to better understand and communicate the limitations of the current blockchain technology. It also helps to explore the guaranteed execution of a "digital doppelgänger," and the strong integrity guarantees offered by a blockchain solution.

3 Role of ACTUS

3.1 ACTUS objectives and contract types

By creating an algorithmic representation of financial contracts, ACTUS enables the dynamic computation (and simulation) of contract properties [2]. It is in essence a set of modelling tools augmented with a data taxonomy; this taxonomy allows for semantic interoperability between the different modelled contracts.

ACTUS defines a set of "*Contract types"* (CT) that can be used to express around 30 different types of financial instruments. ACTUS allows to model the financial state and to describe the cash-flows of these instruments ex-ante. For all contract types, a corresponding set of "*contract attributes*" and "*contract state variables*" are defined. When a specific contract type is instantiated, these attributes and variables need to be initialized.

Contract attributes represent legal contractual terms; these are fixed labels within the model, expressing terms that have a legal interpretation, for example, the "*Initial Exchange Date (IED)*" or "*Maturity Date (MD).*"

Contract state variables, as their name implies, are used to capture contract state. Example state variables are: *Nominal value of the contract (Nvl)*, and *Nominal rate of the contract (Nrt)*. The contract state variables are modified by "*state transition functions*," these are logical rules that are executed when a certain "*contract event*" is applied. Contract events can be seen as logical conditions: when the contract state reflects a certain condition to be true, an event is generated, the contract's logical rules will be executed and the next state of the contract will be reached.

3.2 Limitations of ACTUS

3.2.1 ACTUS is focussed on modelling and simulation

The current ACTUS model and its implementation are focussed on providing a modelling and simulation tool that only takes inputs during the initial model calibration. It is not intended to model *real* world trading and reporting, tracking real data and events. Contract instances cannot be changed in ACTUS once they have been initialized. For example, when party A and party B conclude a basic loan contract – defined in ACTUS as a *Principle at Maturity* (*PAM)* – ACTUS defines a set of state variables that express start and end date of the contract. But ACTUS does not have the concept of an interface to signal during the life time of the contract that a payment between party A and party B has happened. In a simulation, ACTUS is driven by a timing oracle[7] that simulates inputs over time: when a new time event is received ACTUS logical rules determine if a payment event needs to happen. If so, it generates an event and modifies the state. This event is defined in the model, but the payment function is not.
The orientation towards simulation is also illustrated by the *input* state variable *Probability of Default (Pod)*. This is a variable that has no counterpart in a real world contract, it is a variable used during risk analysis. Furthermore ACTUS is probabilities-based. The above limitations were overcome by creating our own implementation of the "digital doppelgängers" in Solidity[8] using ACTUS semantics.

3.4 Our use of smart contracts and of ACTUS

3.4.1 Smart Contracts

In our implementation, smart contracts are used in an unusual way. They are not employed to bind parties to an agreement, or to automate payment in crypto currency. Instead we are using them as trustable, committed bits of calculation, to arrive at results that are signed off and vouched for by the parties who originally staged the information and formulas they rest upon. We are not depending on the guaranteed execution that smart contracts offer.

3.4.2 "HAS" versus "SHOULD" interpretation

During our analysis it became apparent that there were two possible ways of interpreting ACTUS semantics in a smart contract setting. This is referred to as the "HAS" interpretation and the "SHOULD" interpretation. A smart contract could use ACTUS events to signal to the outside world that a certain financial action *should* be taken by the counterparties (this based on the smart contract's current state and a time oracle), this is referred to as the "SHOULD" interpretation. Alternatively a smart contract could use ACTUS semantics to capture the fact that a financial transaction *has* happened, by having an off chain

[6] Our use case encompasses both supervisory and financial reporting.
[7] In a blockchain, *oracles* are defined as sources of information from the world outside the blockchain, as seen from the blockchain, e.g. a smart contract,

[8] The programming language of Ethereum.

source triggering a state change. This is referred to as the "HAS" interpretation. For the demonstrator we chose to implement the "HAS" interpretation,

3.4.3 Use of data oracles in the demonstrator

In order to calculate the correct payment variables, various types of information are required. ACTUS is oriented towards the calculation of expected cash-flows based on market data. In the context of a simulation this data can be provided by e.g. Bloomberg or Thomson-Reuters. Such sources could act as data oracles. They exist but we are demonstrating a fictional case. Therefore, in the demonstrator such market data was 'hardcoded'.

3.4.4 Management of time

Our smart contracts are driven by external time events that move the contract forward in time. We used a strong simplification, where every time event represents a single quarter of a year. The smart contract will execute the logical rules as many times as the *cycle of interest payment (IPCL)* dictates. For example, if the IPCL is initialized to "M" (month) then the corresponding logical rules will be executed three times.

4 Demonstrator

4.1 Overview

The demonstrator shows how digital representations of financial instruments can be implemented as smart contracts on an Ethereum private blockchain and illustrates a core benefit of the blockchain, i.e. consensus across potentially thousands of nodes. On this private chain, transactions and compliance reporting are implemented. Users transact through the various graphical user interfaces (GUI). Their actions are reflected immediately in the state of the individual contracts across all nodes, through the underlying distribution mechanisms of the blockchain. This allows reporting on any contract or set of contracts immediately, at any point in time. And while the contracts' states are changed through the transactions, their storage on a blockchain also guarantees the proper ordering of transactions, and the integrity of results. Furthermore, by sequentially reading all transactions as reflected on the blockchain, the full transaction history is documented. Data governance is implemented by allocating different roles to different participants.

ACTUS seems to be a natural fit with the Ethereum smart contracts, since a smart contract consists of execution state (contract state) and a set of instructions (logical rules), similar to an ACTUS contract that consists of logical rules ("*transition functions*") and contract data ("*contract state variables*", "*contract attributes*").

Two ACTUS contract types are used. The *Principle At Maturity (PAM)* can represent contracts where payment is due at the end of the contract ('at maturity'), e.g. bonds. The *PlainVanillaSwap (PVSWAP)* can represent most swaps: agreements between two parties to exchange cash flows in the future, at specific dates, calculated in a pre-specified way, and over a specific period. We use it to represent *Interest Rate Swaps (IRS).*

Regarding the produced reports, we used representative but simplified versions of each report type mentioned.

4.2 Stories

The first demonstration story addresses bond (PAM) trading where Alice lends money to Bob and later merges with Eve to form the new party AliceEve. The individual steps are the following. Alice creates a new bond contract with Bob. For illustration purposes, the consensus is shown (i.e. the contract is now on the blockchain in every node). Bob inspects the contract details on-chain. The Narrator moves the time to the next quarter. Romeo generates a Common Reporting (COREP) report, as mandated by the European Banking Authority (EBA) for Capital Requirements Directive (CRD) reporting. Such a report covers credit risk, market risk, operational risk, own funds and capital adequacy ratio. The parties Alice and Eve merge (the contract owner is updated). Romeo inspects the contract list and notices the update.

The second story addresses IRS trading, where Bob creates a contract towards AliceEve. The individual steps are the following. Bob creates a new PVSWAP contract with AliceEve. Consensus is achieved between all nodes with regards to the new contract, and is visualised for illustration purposes. The Narrator moves the time to the next quarter. AliceEve looks at the contract details, including calculated fields. Romeo generates an EMIR[9] report on bilateral trades involving derivatives.

The third story covers IRS defaulting, where AliceEve does no longer meet her requirements towards Bob and stops her payments to him. The individual steps are the following. The Narrator generates a default (i.e. indicates a payment stop) on the PVSWAP contract. The Narrator then moves the time to the next quarter, and subsequently inspects the updated state. Finally, Romeo generates a MiFIR[10] report.

5 Technical implementation

The demonstrator is based on a peer-to-peer (P2P)

[9] Derivatives (EMIR) - Regulation (EU) No 648/2012.

[10] Markets in Financial Instruments (MiFIR) - Regulation (EU) No 600/2014.

network, formed by a network hub connecting Ethereum nodes. There are two types of nodes. The first and most basic type is based on Raspberry PIs, while the second type is based on Linux laptops. All nodes participate in the P2P network and run an Ethereum Virtual Machine(EVM), in our case a *geth* client. This *geth* client manages the local copy of the blockchain on each node, where the "digital doppelgängers" are created as smart contracts. The *geth* client also includes a purpose-made consensus-visualization function.

The Raspberry PI nodes do not mine[11] (due to a software limitation in the mining algorithm), while the laptops do mine. The laptops are also equipped with Meteor, a JS application platform that serves the User Interfaces AliceUI, BobUI, RegulatorUI and NarratorUI. The laptops also pull and display the consensus visualization user interface.

6 Limitations of the technology

6.1 Real-world interaction limitations

Current blockchain platforms generally don't allow to interact easily with external systems and the current Ethereum implementation cannot directly fetch data from an external source. This makes it difficult for the "digital doppelgängers" to interact with the real world. Either the simulated world needs to include all required elements (which would make it very complex), or it needs to have a close relationship with the real world, through so-called *oracles*. Two options exist, the first is to pull information from the smart contract by indirectly triggering an exposed function, the second is to have each smart contract monitor all incoming events and act only on the events that are relevant to it.

The Solidity programming language in which we implemented the "doppelgängers" also posed an unexpected issue. At the time of writing it was not possible to declare single or double point floating values (IEEE-756) in the language, and it does not support floating point calculations. This makes it difficult to performing floating point computations (evaluate simple fractions).

We solved this problem by expressing all numerical values as a tuple where the first element is the numerator and the second element is the denominator. For specific calculations we used a scalar value before performing a divide operation.

[11] With blockchains, *mining* is the essential activity that advances the common state across all nodes. It derives its name from the fact that it consists of work to find a proof and is rewarded by the mining fee. But its essential function is to help determining which node will be the leader to propose the next group of ordered transactions, i.e. the next block in the chain.

6.2 Performance and scaling limitations

Classic blockchains such as Bitcoin or Ethereum have strong scalability and performance limits. Smart contracts are around ten orders of magnitudes slower than logic implemented on a Java platform, writing to a blockchain takes three to five orders of magnitudes longer than writing to a database and the throughput is about 5 orders of magnitudes lower. In numbers, Ethereum performs about 25 transactions per second, with each needing about 2 minutes for confirmation. The net data stored in all of the Ethereum mainnet is in the Gigabyte range. Alternative approaches that offer better performance usually forgo the capability to have thousands of nodes, or to support smart contracts. A detailed technical research into twenty different blockchain projects [5] did not yield a fit for the requirements of an implementation of the "digital doppelgängers." The demonstrator visualizes the inner workings that make the technology hard to scale.

One approach to overcome the limitations of current blockchains that can work for the specific supervisory reporting tasks as discussed, is proposed in the paper "Supervisory Reporting Blockchain Architecture" [6] that was prepared in the context of the same study as this demonstrator. However, these proposals are not explored in the demonstrator, to the contrary, in the interest of time we are using a sped up version of Ethereum that would not work for large scale networks.

7 Consensus and synchronization

7.1 Introduction

The demonstrator allows to *visually observe consensus building* between the individual nodes in the network in real time. The specific way that the state of thousands of nodes can be synchronized in public blockchains is the major invention of Bitcoin. We are using the *proof-of-work* algorithm that is also employed in Ethereum. Consensus is built between nodes on the basis of the *root hash* of the Merkle tree[12] of the entire state, current and past, of the blockchain. In this way, finding agreement about only 32 bytes can facilitate perfect consensus about gigabytes of data. This process has several steps: collecting transactions, mining, distributing the block and verifying received block proposals. All of these can be witnessed. It revolves around finding a random number, called *nonce*, that gives the right to propose a new block to the network

[12] A Merkle tree is an efficient way to hash large datasets. Hashes work like checksums that make it easy to detect if a large dataset was changed or not. They are functions that are easy to calculate in one direction – from the data to the hash – but hard to reverse.

and earn the mining reward. Mining is the search for the nonce, which requires a lot of calculations.

7.2 Visualization Device

The visualization consists of one dynamic web page per node that shows, in real time, what the internal, partially transient state of the blockchain client on that particular node is. Large, coloured letters serve as symbols for the hashes that the consensus is formed about, as well as other relevant state in the client programs. *Whenever two pages display the same colour letter in the same box, it means that the two clients that the pages represent are in consensus about this data point.* The most relevant consensus is about the root hash of the blockchain itself, which is what is generally meant when talking about consensus in the context of blockchains. The other letters provide insight into the individual steps of the process.

7.3 Shown Data Points

Each of the following data points are updated live in the web pages for each individual client:

Tx Broadcast
Hash of the last transaction that this node has broadcast to the network, i.e. to any of its peers.

Tx Received
Hash of the last transaction that this node has received from the network, i.e. from any of its peers.

Work: Nonce Tried
Last nonce that this node has tried out to form a block. This is a fast, continuous action when this node is mining, and none if not. Nonces are tried by the millions per second and the display will show some snapshots per second.

Proof of Work: Nonce Found
Last nonce that this node found to successfully form a block. This nonce *is* a valid "*proof of work*." This is a slow action that can occur around every 5 to 60 seconds when this node is mining in a small private network, none if not.

Proof Accepted: Nonce verified
Accepted proof of work from another node: the last nonce that this node found to match a block as expected, coming from a peer. This nonce has been verified to be a valid "proof of work" that a peer found and showed.

Proposed Block
Hash of the last block that this node proposed to the network. Slow action around every 5 to 60 seconds when this node is mining in a small private network, none if not.

Root Hash
Hash of the highest, and last block that this node accepts as top of the blockchain. This is 'the' block hash that is *chained* to the next block and that all participating nodes form consensus over.

Number of Peers
Number of peers that this node has and communicates with. The node does not communicate with all peers all of the time.

Chain Height
Number of highest, and last, block that this node accepts as top of the blockchain, as counted in unbroken line from the genesis block. This is 'the' height[13] of the chain that (mostly) decides which version of the blockchain prevails when block proposals compete or a partition is re-united.

7.4 Observable Rhythm

The rhythm that is observable is that of one client leading and the others following within a couple of seconds. This is the expected behavior, showing how one node takes the lead and proposes a block and how the remainder of the network picks the block up, validates the data in it and eventually *implicitly* agrees on. When it is internally accepted as part of the current state, the letter on the display changes to the represent the new root hash of the blockchain, now including the new block. After a short while all nodes show the same colour letter again. This demonstrates how nodes lose and find consensus again with every state transition of the network.

There are also patterns of 'rogue' blocks observable, where for a short time some clients display a different colour letter, symbolizing a different block's hash, which shows that a short lived proposal is being accepted only to soon be overwritten by a proposal that receives more support from the rest of the network.

7.5 Technical Setup

The visualization uses a special, modified version of the Ethereum *geth* client that provides access to specific internal variables that the Ethereum API does not provide, at a controllable frequency and pushing the data out from the client. Each client runs a primitive web server that is used to create the pages with coloured letters. The dynamic rendering happens mostly in the browser, using Javascript. Data is pulled by the browser from the web server. The modification also speeds up block production of Ethereum for the sake of a more fluent demonstration. The modifications to *geth* are minimal and there are no changes to core blockchain functionality.

[13] This is a simplification that holds true for Bitcoin but works in a more complex way in Ethereum, where 'weight' is defined for blocks.

7.6 Detailed information and Source Code

More detailed information about the meaning of the individual data points, their expectable frequency of change and consensus, as well as how to set up such a demonstration and the source code can be found at:
https://github.com/claryon/vizmod

Acknowledgements

We thank Wolfgang Breymann and Petros Kavassalis for the interesting discussions and clarifications regarding ACTUS. We thank Peter van den Hul for his guidance and support during the project.

References

[1] Kavassalis, P., Saxton, K., Gross, F., Agarwal, P., Stieber, H. (2016), "Financial data for the Good Society: an innovative data-science approach for Big Data regulatory reporting", accepted paper 61 at "Frontiers of Data Science for Government: Ideas, Practices, and Projections", 15-16 September 2016, University of Cambridge, Cambridge, UK.

[2] ACTUS Financial Research Foundation (2017), "ACTUS High Level Documentation", Version 0.2, Date of version 2017-04-06, available at: http://actusfrf.org/index.php/algorithmic-standard/#

[3] Flood, M., Goodenough, O. (2015), "Contract as Automaton: The Computational Representation of Financial Agreements", OFR Working Paper 15-04 | March 26, 2015, Revised March 27, 2017, available at https://www.financialresearch.gov/working-papers/files/OFRwp-2015-04_Contract-as-Automaton-The-Computational-Representation-of-Financial-Agreements.pdf

[4] Zyskind, G., Oz, N., Pentland, A. (2015) "Decentralizing Privacy: Using Blockchain to Protect Personal Data, in Proceedings of the Security and Privacy Workshops (SPW), IEEE, available at http://web.media.mit.edu/~guyzys/data/ZNP15.pdf

[5] Diedrich, H., Becze, M. (2017) "Supervisory Reporting Blockchain Architecture", mimeo

[6] Diedrich, H. (2017) "Blockchain Specification Matrix", mimeo