A Structured Semantic Domain for Smart Contracts
Extended Abstract

Sören Bleikertz, Andreas Lochbihler, Ognjen Marić, Simon Meier, Matthias Schmalz, Ratko G. Veprek
Digital Asset (Switzerland) GmbH, Zurich, Switzerland
firstname.lastname@digitalasset.com

Smart contract languages provide a new paradigm for coding business workflows that span multiple untrusted actors. Their semantics are given in terms of *transactions*, updates to a model of a *ledger* that is shared between the actors. A distributed system then implements this semantic model and provides users with the distributed ledger. Most models follow the ideas of Bitcoin [5] or Ethereum [6]. In Bitcoin, a transaction is described by two sets: the spent immutable coin inputs and the produced immutable coin outputs; the ledger state consists of the unspent outputs (UTXOs). In Ethereum, the ledger state, so-called "world state", maps addresses to values, and transactions mutate this mapping. Both approaches suffer from confidentiality, privacy, and scalability issues [2], [3].

In this abstract, we sketch how adding more structure to the semantic domain yields a more secure programming model for smart contracts, and allows for distributed implementations with better confidentiality, privacy, and scalability properties.

We identify five distributed ledger desiderata that the UTXO and world state models lack. They are motivated by typical use cases for smart contract platforms like trading assets between multiple actors, e.g., a delivery-versus-payment (DvP) swap in a financial market. Here, Alice transfers an asset A to Bob and Bob transfers an asset B to Alice. The desiderata are:

a) Compositionality: Small ledger updates should be composable into larger updates. For example, given an update for transferring an asset from the owner to another party, we can combine two of them into a DvP update. The composition must be *atomic*: one asset is transferred if and only if the other asset is transferred. Lack of atomicity is a systemic problem for conventional financial systems: when Bob defaults between the two transfers, Alice has already given up her asset without getting anything in return. Intermediaries can take over this risk, but Alice and Bob must trust them to correctly execute the swap on their behalf and to not default while doing so.

b) Authorization: As the ledger is shared between untrusted actors, every ledger update must be properly authorized. For example, an asset owner must authorize the transfer to another party, and both parties must have agreed to the DvP swap. For compositionality, authorization should support delegation of rights. As part of a DvP contract, e.g., Alice can delegate to Bob her right to transfer her asset to Bob, so that Bob can execute the swap without involving Alice. When the digital assets represent off-ledger obligations, authorization is crucial in ensuring that these obligations are entered into voluntarily.

c) Confidentiality: As ledger updates may contain sensitive data such as trade secrets and GDPR-relevant data, they should only be visible to the parties that have a stake in it. For example, an unrelated third party Eve should not learn which assets were exchanged in a swap between Alice and Bob. Ideally, confidentiality can be maintained at the level of sub-updates. That is, if ownership of Alice's asset is tracked by the registry R, then R should be involved only in the transfer of Alice's asset. In particular, R should not learn what is exchanged in the other leg, i.e., what Alice gets in return.

d) Privacy: Not only must the business details remain confidential. Also the parties involved in one sub-update should remain hidden from parties involved only in unrelated sub-updates. For example, the registry R need not learn who is the registry in the other leg.

e) Scalability: Scaling needs sharding and parallel processing. A ledger based on a global shared state limits scaling.

Contributions. We propose a new semantic domain [4] for distributed ledgers that achieve all of the above desiderata. At its core are a hierarchical transaction structure and a notion of data and action ownership. The transaction structure is given by the following Haskell types, where we leave abstract the types `Party` for parties and `Contract` for smart contracts.

```
data Action
  = Create           Contract
  | Exercise [Party] Contract Kind Transaction
  | Fetch    [Party] Contract
data Kind = Consuming | Nonconsuming
type Transaction = [Action]
type Update = ([Party], Transaction)
type Ledger = [Update]
```

The basic building block `Action` captures a ledger change. There are three kinds: First, `Create` creates a smart contract. Second, `Exercise` records that the given parties, called the actors, have exercised a right on the given contract. The `Kind` determines whether the `Exercise` consumes the contract; consumed contracts cannot be used afterwards. The last argument describes the consequences of the exercise as a subtransaction, i.e., a list of actions. Third, `Fetch` checks whether the contract is active, i.e., it has been created and not yet consumed; the parties are the actors that authorize the check.

An `Update` consists of the requesting parties and the actual (atomic) change as an `Action` list. A `Ledger` lists `Update`s.

Semantically, a smart contract is modelled by (1) a predicate `valid`, which determines whether an action on the contract is valid, and (2) sets of contract *signatories* and *observers*. Signatories "own" the contract data and are bound by the

contract. Signatories and observers are privy to actions on the contract. Contracts are immutable like in the UTXO model. So when the details of a smart contract should be modified, e.g., the owner in a contract modelling asset ownership, the contract is consumed in an `Exercise` action whose consequence creates the updated instance as a new contract.

The structured ledger updates allow us to express and reason about security properties like authorization, confidentiality, and privacy within our model. Reflecting this model in our smart contract language DAML (https://www.daml.com) is how we achieve the stated desiderata.

The hierarchical structure of actions gives us compositionality, as transactions are atomic. An asset transfer from Alice to Bob, e.g., is expressed by `Exercise [Alice] asset Consuming [Create ...]`, where ... describes the creation of an asset contract for the owner Bob. This action can be one of the consequences of an `Exercise` on a DvP contract.

Authorization is achieved as follows. Every `Exercise` and `Fetch` action must be authorized by its actors. Every `Create` action must be authorized by its signatories. An update is authorized by the parties requesting it. Furthermore, a consequence of an `Exercise` is jointly authorized by the actors together with the signatories of the exercised contract. This authorization rule enables delegation.

Confidentiality and privacy are captured by projections, which determine which parts of a ledger update a party may see. Roughly speaking, projecting an action to a party yields the list of subactions where the party is an actor or a signatory or observer of the affected contract. So if a party does not appear in a ledger update, then it sees nothing of the update. And if it is involved only in a few subactions, then it sees only those subactions and their children. This enables scaling, as a party does not store or process data that it does not see, but disallows a few use cases, e.g., bearer tokens.

This ledger model is not only theoretical. The semantics of our programming language DAML are defined in terms of this model. The Australian Stock Exchange is set to deploy a DAML implementation of trade clearing and settlement (C&S) to production in 2021 [1]; C&S is a core part of the stock exchange business. Furthermore, we have designed and are implementing a synchronization protocol, CANTON, which implements this ledger as a scalable distributed system.

In CANTON, parties deploy CANTON *participants* (Fig. 1). Participants run the CANTON protocol and communicate via *synchronization domains*, which order messages. A participant can connect to multiple domains simultaneously, and processes messages from different domains in parallel. A ledger update can be processed whenever all involved participants are connected to some joint domain. CANTON ensures the *integrity* of the shared ledger, defined as follows: (1) every contract is created at most once; (2) all contract are active when fetched or exercised (to prevent double spends); (3) all ledger actions satisfy `valid`; and (4) all actions are authorized. To achieve integrity, confidentiality, and privacy simultaneously, CANTON relies on the following fundamental property of our model, which holds under mild conditions on the predicate `valid`.

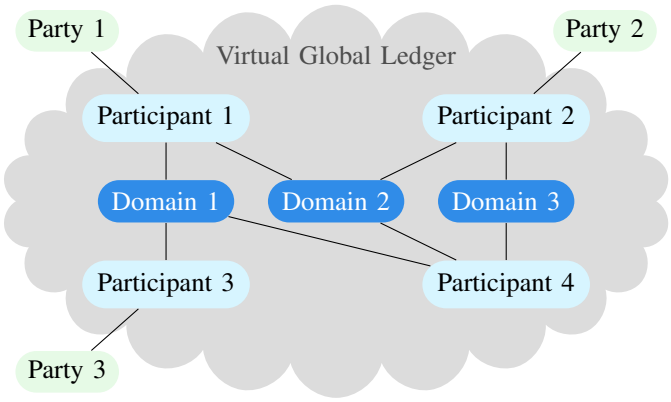

Fig. 1. The CANTON architecture

Distributed Ledger Propertyx A ledger satisfies integrity iff the projections to each party do so.

This property enables CANTON participants to perform the integrity checks only on their projections of an update, and yet achieve integrity for the shared ledger. Stating the achieved integrity property is not trivial, as we allow both honest and Byzantine participants. The formulation is as follows: CANTON ensures that there exists a shared ledger that satisfies integrity, such that its projection to each honest participant (i.e., their party) consists exactly of the ledger updates which have passed the participant's local checks. CANTON achieves this property under the following modest trust assumptions:

- The domain correctly implements total-order multicast for messages sent over the domain.
- A special entity in the domain, the *mediator*, correctly aggregates the outcomes of the participant's local checks, like a coordinator in a two-phase commit. It hides the participant's identities and thereby ensures privacy.

The actual ledger updates are encrypted so that the domain entities cannot see the contents. Moreover, every non-liveness violation of integrity can be attributed to dishonesty or faultiness.

We have formalized our semantic domain and the distributed ledger property in the proof assistant Isabelle/HOL. Now, we are working on formally verifying CANTON's integrity guarantees using Isabelle/HOL.

Acknowledgements: We thank all current and former Digital Asset employees who worked on DAML and CANTON. The authors are listed alphabetically.

REFERENCES

[1] ASX Chess Replacement. https://www.asx.com.au/services/chess-replacement.htm, version 2019-04-29.
[2] N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on Ethereum smart contracts (SoK). In *POST 2017*, pages 164–186. Springer, 2017.
[3] M. Conti, E. S. Kumar, C. Lal, and S. Ruj. A survey on security and privacy issues of Bitcoin. *IEEE Commun. Surveys Tuts.*, 20(4):3416–3452, 2018.
[4] DA ledger model. https://docs.daml.com/concepts/ledger-model/index.html, version 2019-04-16, 2019.
[5] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.
[6] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.