Architecture for Smart SAFE Contracts

Ron van der Meyden
UNSW Sydney

Michael J. Maher
Reasoning Research Institute

Abstract—**This paper proposes an architecture for implementing Y Combinator's Simple Agreements for Future Equity (SAFEs), a class of financial instruments used in funding startups, as smart contracts. We describe design patterns used and properties of the architecture that combine to ensure correctness of the Solidity implementation.**

I. Introduction

Direct sale of shares is just one form of financing used by companies: others are loans and more complex "convertible" instruments that mix loans and equity. The present paper is part of a larger project on the applicability of smart contract technology to this more complex form of financing contract, via a case study of a simple form of convertible instrument, Y Combinator's Simple Agreement for Future Equity (SAFE) [Y C16], [Y C18]. These are a form of contract in which an investor invests a sum of money in a company in exchange for a promise of shares to be issued at the time of a future priced equity round. Valuation of early stage ventures can be difficult given their unproven and speculative status. The contract enables the investor to defer the question of valuation of the company, allowing this (and hence a price for the shares) to be set at the time of the future round.

Although a SAFE contract itself is short, at just 6 pages of text, developing a smart contract representation, in addition to the technicalities of design, coding and verification, requires addressing many difficulties: legal interpretation, the fact that these contracts are performed in variant ways in practice, the need to deal with "open textured" language in the legal text, consideration of privacy concerns related to contract data, the need to be able to accommodate legal rulings. We have addressed some of these issues elsewhere [vdMM20], [vdMM21], and focus in the present paper on the question of software architecture for this application. Amongst the specific challenges of the application is the *open* nature of what needs to be protected using the DLT platform: the financial contracts we aim to support come in myriad forms, which are subject to negotiation with the investors. Similarly, the governance structures imposed on the company (its board rules, voting rules, and constraints on its officers) as it matures, are open ended. We develop an approach to representation of the scenario that enables flexibility in both the financial contracts that the company issues and the governance structures adopted.

A further issue, both in analysis of the application and its implementation on a blockchain, is the potential for interference between events captured by a single transaction, and processes (such as an equity round) that are "long-running", requiring several transactions initiated by different agents. We resolve this by serialization using a state-machine that captures states in which a long-running process is in progress.

We apply a principle of systems architecture, particularly in the context of secure systems development, which is to isolate critical functionality in components that are small enough to be verified (ideally, using formal methods), combined in the architecture in ways that ensure that the basis supporting key properties remains small. We apply this methodology, representing a company that issues SAFE contracts using a collection of smart contract modules, each of which ensures some key properties of the overall application.

Key to the architecture are a number of known smart contract patterns: the *owner* pattern and the *atomic swap* pattern. These are described in Section III. We show how these patterns support the open ended nature of the application, and can be combined to enforce a key clause of SAFE contracts in Section IV. Section V concludes with a brief discussion of related work.

II. Summary of SAFE Contract Structure

SAFE contracts come in a number of forms. As initially issued by Y Combinator, in a form we call "Pre-money SAFE contracts", these contracts have two parameters, a "Valuation Cap" and a "Discount", that may or may not be included, giving four distinct contracts. The Valuation Cap gives a maximum valuation used to calculate a price at which the SAFE principal will be converted to shares, and the Discount relates to a percentage discount given on the price. SAFE contracts describe the investor's rights in case of three types of events: Equity Financing, Liquidity (e.g. sale of the company), and Dissolution (winding up of the company).

The key clause of the SAFE that we focus on in the present paper is the following, from the Events section, in the case of the Pre-money SAFE with cap and no discount [Y C16]:

> **Equity Financing.** If there is an Equity Financing before the expiration or termination of this instrument, the Company will automatically issue to the Investor either: (1) a number of shares of Standard Preferred Stock equal to the Purchase Amount divided by the price per share of the Standard Preferred Stock, if the pre-money valuation is less than or equal to the Valuation Cap; or (2) a number of shares of Safe Preferred Stock equal to the Purchase Amount divided by the Safe Price, if the pre-money valuation is greater than the Valuation Cap.

The "price per share of the Standard Preferred Stock" is treated as a primitive input available at the time of the equity round, and the "Valuation Cap" is a parameter that is filled in when instantiating the contract before signing. The "Safe Price" is defined from the Valuation Cap and the number of shares and convertible instruments issued.

Y Combinator varied their standard contracts in September 2018, introducing a "Post-Money" version of the SAFEs. The overall structure of these documents is as before, but there are substantial changes in the definitions [Y C18]. A significant issue that this raises is that, whereas the Pre-Money SAFE gives an explicit definition of the number of shares to be issued (depending on some undefined terms), the Post-Money SAFE has a circular definition, that can be understood as stating a set of constraints that need to be satisfied in order to determine the shares to be issued to a SAFE investor. (See [vdMM20] for detailed discussion of the constraints and their solution.)

III. Design Patterns

A number of general design patterns prove to be useful in the development of chaincode representations of SAFE contracts. In this section we give a general introduction to two of these patterns.

A. Atomic Swap

Suppose that two parties A and B have agreed to exchange digital assets a and b, respectively. If the transfers are made sequentially (e.g., first A transfers a to B and B transfers b to A), each party faces the risk that the other party will not abide by their end of the bargain (e.g., B could receive a but then refuse to transfer b to A, leaving A with a loss). Smart contracts provide a method to overcome this difficulty, in the form of *atomic swap* transactions [Her18], [vdM19].

Atomic swaps can involve multiple blockchains, but for our purposes in this paper, swaps on a single chain will suffice. These can be effected by chaincode having the following structure, in the case of a swap of asset a held by party A for an asset b held by party B. There are operations *Deposit* whereby the parties can transfer their assets to the control of the smart contract, so that the assets come under the smart contract's control. Once both assets have been transferred, the operation *Swap* transfers each asset back to the opposite party. In case the other party fails to perform their *Deposit* action in a timely fashion, the action *Withdraw* can be used to recover an asset that has already been deposited, so that the swap then fails. The effect is that the swap happens atomically, either taking full effect, or leaving both parties with effective control over their original asset (i.e., either holding it, or able to retrieve it from the swap contract).

Usually such swaps are asset for asset (e.g., one form of cryptocurrency for another), but richer forms of swap smart contract can be constructed (e.g., a multi-party contribution of cryptocurrency or approval in exchange for a change of state in chaincode). We use such a richer form of swap below to secure aspects of equity rounds.

B. Controller

A common form of access control in object oriented programming is that some operations may be performed only by one specific agent. In Solidity code, this is captured by the *controller* (or *owner*) pattern (see, e.g., [Ope]). In Solidity, this may be done concisely by a modifier that checks that the caller is equal to the address in a variable `controller` .

Typically, the value of the `controller` variable is the address of the public key of a human agent. However, we note that once a smart contract has been created with the controller pattern, it is possible to impose richer forms of access control by taking the value of `controller` to be the address of a *controller smart contract* that encodes the logic of the richer form of access control. We use this *constrained controller* pattern in the architecture presented below.

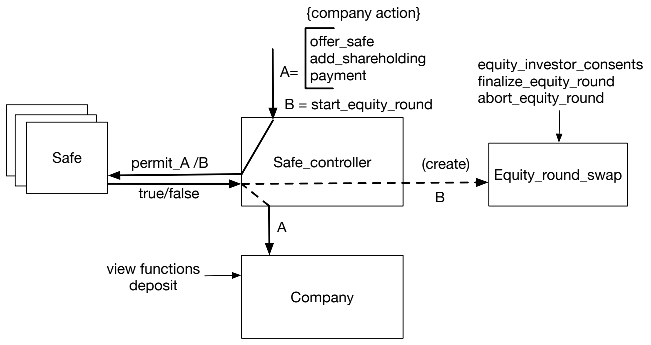

Fig. 1. Architecture with SAFE_controller

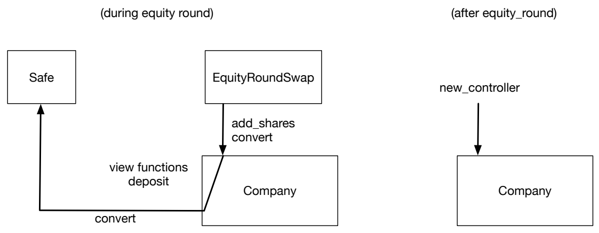

Fig. 2. Architecture configuration during an equity round

IV. Architecture and Implementation

Factors, noted above, that influence the design of our smart contract solution for SAFE contracts, are the *open structure* of the situation we are modelling, and the issue that Post-Money SAFE contracts are stated as a *constraint* on the promised shares rather by a method to compute their number. To address that there are multiple forms of SAFE contract, we have abstracted these to an abstract contract (instantiated by subclassing) that places constraints on the Company and the way it manages an Equity round. To handle the openness and dynamic nature of the governance structures of the company over time, we use the constrained controller pattern.

To address the fact that Post-money SAFEs state but do not solve a constraint system on the equity round, rather than have the smart contract *calculate* the number of shares to be issued to SAFE investor, we have the company *propose* the structure of an equity round, and have this proposal *verified* for correctness by the smart contract.

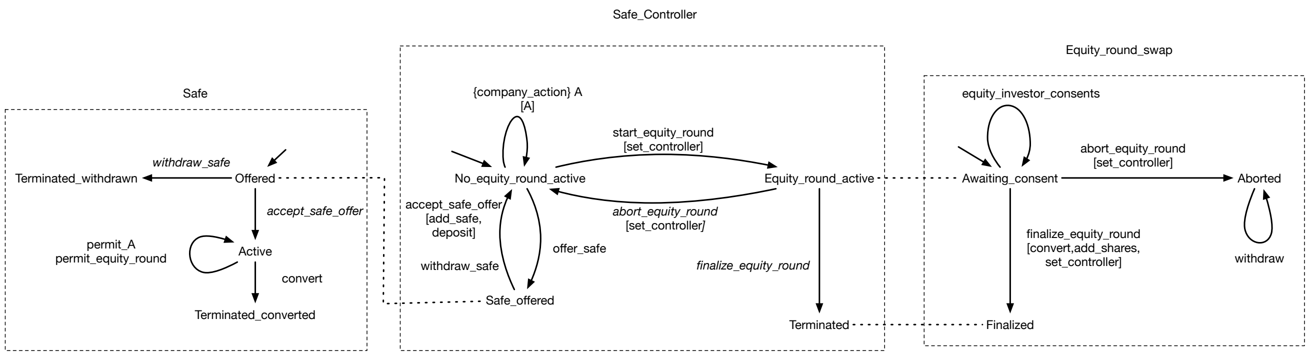

Fig. 3. Contract State Transitions

The architecture of our solution is dynamic, with its principal configurations shown in Figure 1 and Figure 2. Figure 3 depicts the finite state-machine view of some of the components, that underpins some key correctness properties.

A. Company Financial Structure

Our on-chain representation of the state of a company will be a smart contract of class `Company`, that

- may hold cryptocurrency, representing some of the assets of the company, and
- represents the cap table of the company, in the form of a record of shares held by each investor in the company,
- represents the convertible instruments issued by the company (assumed to be SAFE contracts only).

It should be possible to update these variables by means of various operations, such as

- `add_shareholding` to issue new shares,
- `add_safe` to issue a new SAFE
- `payment`, which transfers some of company's cryptocurrency holdings, to pay for goods and services
- `deposit`, used by other parties to transfer cryptocurrency to the company

However, these operations are not unrestricted: only a duly authorized officer of the company should be able to issue new shares or SAFEs, and there are circumstances in which the company will be constrained in performing such actions. For example, equity round term sheets, i.e., preliminary contracts setting out the expected conditions of an equity round as a basis for more detailed negotiations, may constrain the company from changing its capital structure until closing or abandonment of the equity round. After an equity round, the governance structure of the company is likely to have changed, e.g., by instituting a board with membership from the new investors, and issuance of new stock will typically require approval of the board and/or a majority vote of shareholders.

To capture such constraints, we therefore require a means of access control on the performance of actions on the cap table. This can be done in an abstract way using the controller pattern from Section III-B. We take the original controller to be the agent (address) creating the `Company` smart contract, and include a function `set_controller` whereby the controller (and only the controller) may transfer control to another agent (possibly a smart contract).

B. Safe Contract Representation

SAFEs come in multiple forms: Pre-Money SAFEs and Post-Money SAFEs, each of which may include a cap and/or a discount. We develop a representation that can accommodate all of these types of SAFE. Our approach allows a company to issue a set of SAFE contracts of variant types. We achieve generality by defining an abstract smart contract class `Safe`, which may be subclassed to instantiate specific types of SAFE. The main variables of this abstract class represent the amount paid for the SAFE (in Wei), the party holding the SAFE, the smart contract for the company being invested in, and the status of the SAFE, a value from the Enumerated type {`Offered`, `Active`, `Terminated_withdrawn`, `Terminated_converted`}. A `Safe` is offered to the investor in state `Offered` and becomes `Active` once accepted with payment of principal. A deadline is given for acceptance.

Once a company has issued a SAFE, it needs to comply with restrictions imposed by the SAFE. In the `Safe` smart contract itself, we capture the logic of these constraints, but do not actually enforce them. Enforcement will be done using the controller pattern, by setting a controller for the company that queries the `Safe` contracts to determine which actions on the company are consistent with all the SAFE contracts that the company has issued.

The logic of constraints that the SAFE places on the company is captured by functions in the `Safe` smart contract named `permit_A`, where `A` is an action (function) on the `Company`. The parameters of `permit_A` and `A` are the same. The function `permit_A` may query the state of the contracts at `company` and `safe_controller` and returns a boolean value, which expresses whether the company is permitted to perform action `A` in that state. These functions will in fact be called only when no equity round is active, and no SAFE is presently on offer, because of the serialization policy described below, so we focus on that case.

Approval of an equity round is obtained by calling a function `permit_equity_round` on each `Safe` that the company has issued. The parameters of the function give a detailed description of the structure of a proposed equity round[1]. To support automated enforcement of the SAFE holder's interest, the function should verify that that the proposed equity round is consistent with the terms of the "Equity Financing" clause of the SAFE.

If the equity round does eventually successfully complete, it is only then that shares are issued to the SAFE investors in conversion of their SAFEs. To signal to the `Safe` smart contract that the conversion has been performed, the `Company` calls a function `convert()` in the `Safe` contract, which sets variable `status` from `Active` to `Terminated_converted`.

C. SAFE controller

The purpose of the `Safe_controller` smart contract is to act as the controller of a `Company`, in order to ensure that the company is compliant with all SAFEs it has issued. Before issuing a SAFE, a company should set its controller to be an instance of a `Safe_controller` smart contract and, to ensure that their interests will be protected by the code, a SAFE investor should not invest in a `Safe` unless a `Safe_controller` is in place.

The `Safe_controller` serializes the permitted company actions with the compound processes of issuing a SAFE and conducting an equity round. This is achieved by means of state variable with values in enumerated type

```
{No_equity_round_active, Equity_round_active,
 Safe_offered,Terminated}
```

and transitions as depicted in Figure 3. Functions are enabled only at states from which they label an outgoing transition. Operations on the `Company` contract called by these functions are indicated in square brackets. Dashed lines between states in different smart contracts indicates that transitions to these states occur simultaneously in response to some action.

D. Atomic Swap Implementation of the Equity Round

An atomicity issue similar to that solved by the atomic swap of assets between two parties (see Section III-A) arises at the time of the equity round, but involving a larger number of parties (the Company, the SAFE investors, and the equity round investors). Each would like to be assured that they will be issued the number of shares negotiated, or due from a SAFE, and that variations from the agreement will not alter their expected relative stake. Suppose that the parties with negotiation rights come to agreement on the number and type of shares to be issued to each of the SAFE and equity round investors. We can provide assurance to all parties that the equity round will be conducted in accordance with this agreement by codifying it in an atomic swap-like smart contract that first collects (where required) the signed consent of each the investors, as well as the new investment moneys, and then issues shares to all of the investors as per the agreement. In case some of the required consents or moneys are not obtained by a deadline, the contract aborts and the moneys can be recovered by the respective investors.

The `Equity_round_swap` smart contract in the implementation realizes this idea. It has local variables that encode the details of the proposed equity round. A `status` variable of enumerated type `{Awaiting_consent,Finalized,Aborted}` and initial value `Awaiting_consent` records the current state of equity round workflow.

[1]The parameters are the price of each share, the pre-money valuation, the investor, principal paid, and class of shares to be issued for each of the new investors, the number of shares to issued to each of the SAFE investors in conversion of their shares and the type of shares to be issued to each SAFE investor, the address of the controller of the company after the equity round completes, and a deadline for completion of the round.

V. Related Work

Multiple commercial efforts (e.g., by ASX, Vertalo, MyStake, CoreConX) are presently underway to provide blockchain-based cap table management services, but they do not appear to support SAFEs. Simple Agreements for Future Tokens (SAFT) [BBSC17] was a legal attempt to base ICO's on the SAFE model. To our knowledge, SAFTs have been issued only as legal rather than as smart contracts. OpenLaw has used SAFEs in a demonstration video (https://www.youtube.com/watch?v=ySIsxEl5yME) relating to launching smart contracts from a template-based legal contract platform, but the smart contract code does not appear to capture the contract logic to the depth we have attempted here. They have also released a contract for an interest bearing convertible bond with an option to convert to tokens, but the online tool for this generates only text (https://consensys.net/convertible-note/).

Other related work includes [GIM+18], which develops a smart contract from a legal contract. But it only addresses an artificial licensing contract and only makes a comparison between imperative and declarative programming approaches to smart contracts.

References

[BBSC17] Juan Batiz-Benet, Marco Santori, and Jesse Clayburgh. The SAFT project: Toward a compliant token sale framework. Online https://saftproject.com/static/SAFT-Project-Whitepaper.pdf, Oct 2017.

[GIM+18] Guido Governatori, Florian Idelberger, Zoran Milosevic, Régis Riveret, Giovanni Sartor, and Xiwei Xu. On legal contracts, imperative and declarative smart contracts, and blockchain systems. *Artif. Intell. Law*, 26(4):377–409, 2018.

[Her18] M. Herlihy. Atomic cross-chain swaps. In *Proc. ACM Symp. on Distributed Computing*, 2018. Version at arXiv:1801.09515.

[Ope] OpenZeppelin. Documentation: Ownership. Online https://docs.openzeppelin.com/contracts/2.x/api/ownership#Ownable. Accessed 25/3/20.

[vdM19] R. van der Meyden. On the specification and verification of atomic swap smart contracts. Online: https://arxiv.org/abs/1811.06099, 2019. Abstract appears in IEEE International Conference on Blockchain and Cryptocurrency, 2019, pp. 176–179.

[vdMM20] R. van der Meyden and M. J. Maher. Simple agreeements for future equity – not so simple? manuscript, http://www.cse.unsw.edu.au/~meyden/research/SAFEnss.pdf, 2020.

[vdMM21] R. van der Meyden and M. J. Maher. Indeterminacy and smart contracts – a case study. submitted, http://www.cse.unsw.edu.au/~meyden/research/SAFE-open.pdf, 2021.

[Y C16] Y Combinator. *Safe: Cap, no Discount*, https://web.archive.org/web/20180831020232/http://www.ycombinator.com/docs/SAFE_Cap.rtf 2016.

[Y C18] Y Combinator. *Safe: Valuation Cap, no Discount*, https://web.archive.org/web/20190626002912/https://www.ycombinator.com/docs/Postmoney\%20Safe\%20-\%20Valuation\%20Cap\%20-\%20v1.0.docx 2018.