2019 IEEE International Conference on Blockchain (Blockchain)

Formal Verification of Smart Contracts using Interface Automata

Gabor Madl, Luis A. D. Bathen, Germán H. Flores, Divyesh Jadav
Cloud System Analytics
IBM Research, Almaden
San Jose, CA 95120
{gmadl, bathen, ghflores, divyesh}@us.ibm.com

***Abstract*—Emerging distributed ledger technologies – also known as blockchains – have recently become a hot research topic across a wide variety of industries due to their immutability, availability, security, and more importantly, their ability to execute multi-party agreements or business rules in the form of smart contracts. Smart contracts provide the means to automate and enforce contractual terms between the parties entering the agreement. It is important to guarantee the correctness of such contracts in order to avoid catastrophic events that may require human intervention. This paper presents a method that builds on the interface automata model of computation as a semantic domain to formalize smart contracts. The formal model of computation assigns a precise and unambiguous meaning to smart contracts. Loyalty points are discounts and bonuses provided by companies with the purpose of retaining their loyal customers. The proposed method is evaluated in the context of a loyalty points marketplace. The decentralized, distributed and immutable nature of blockchain provides an appealing platform for the implementation of a loyalty points marketplace, and the formal semantics based on interface automata ensure that violations of the agreement can be detected, and contractual clauses can be enforced by all parties involved in loyalty points transactions.**

I. Introduction

Blockchain provides a transparent, distributed, decentralized and immutable storage fabric that has become a popular choice to manage transactions between multiple untrusting parties. The first applications of blockchain technology where in the domain of crypto currencies such as Bitcoin and Ether, which demonstrated the viability of the technology as a world-scale financial service [1].

As time progressed, a more complex *smart contract* engine was desired. Ethereum [2] was the first to introduce a Turing-complete smart contracts. The introduction of Turing-completeness meant that smart contracts could now encode more complex business rules. The terms of blockchain and distributed ledger technology were minted and used to denote a blockchain fabric capable of executing smart contracts in a decentralized way. As time passed by, new efforts to create permissioned as well as permissioned blockchains were well on their way [3]. The increased complexity of the smart contract language has led to bugs/vulnerabilities triggered or exploited [4]. The largest to date is known as the *DAO attack*, in which a code vulnerability was exploited to drain 3.6 M ether (15% of all ether in circulation at the time) from its smart contracts [4].

Over time, data recorded in blockchain has evolved from simple transactions of exchanging tokens from one user to another. Smart contracts can capture more complex forms of interaction between potentially multiple parties. Smart contracts provide the means to capture complex agreements, similar to legally binding contracts in the real world, and even games such as Crypto Kitties [5].

A smart contract by definition must be precise, unambiguous, and provide a way to facilitate, enforce, or verify a contractual clause through automated means i.e. through the use of computers. Contractual agreements define the terms by which the parties to the agreement must abide. Contracts are typically written in English legalese, making the limitations of the agreement hard to understand. The primary reason behind this is the lack of *formal semantics* in natural languages. One lawyers understanding of a contract may differ from another, and will likely differ quite a bit from that of the end user.

A formal and detailed approach is necessary to unambigously capture agreements between blockchain users as smart contracts. Section II defines the problem statement and formalizes the interface automata [6] proposed for the verification of smart contracts. By formally modeling the end user requirements and services provided by blockchain service providers, one can formally verify whether the requirements and provided services are *compatible* with each other. Moreover, the process must capture requests and acknowledgements to properly trace the whole *process of the agreement*. This approach provides a way to identify the cause of issues - and the responsible party - when end user requests do not get served. The *composition* of the end user requirements and the service interface results in a smart contract, that captures key aspects of the smart contract in a clear and unambiguous format. Violations between the smart contract and actual implementation can then be detected and enforced by either party to the agreement.

The proposed system and method is demonstrated through a *loyalty points* case study in Section III. Loyalty points are a way through which companies can reward their customers. For example, airlines may provide free miles to certain frequent flyers, hotels can provide free accomodation or discounts to frequent guests, parks and casinos can hand out discounts to visitors, and restaurants and coffee places can provide discounts to people frequenting their establishments. Loyalty

978-1-7281-4693-5/19/$31.00 ©2019 IEEE
DOI 10.1109/Blockchain.2019.00081

points represent actual monetary value to customers. Despite this, the exchange of loyalty points between vendors and users is either cumbersome, non-existent or expressly prohibited by the loyalty points issuer. There is no marketplace mechanism in place to facilitate such exchanges. A central solution likely will not work for such applications. Which company would sign up for such service knowing that it is managed by a single entity who might as well be their competitor?

Section IV describes the proposed approach for the verification of smart contracts. The decentralized, distributed and immutable nature of blockchain provides an appealing platform to implement and manage a marketplace for loyalty points exchange. We integrate smart contracts in the blockchain as the means to handle contractual obligations and disputes between participants. The formal semantics provides a way for unambiguous agreements and is greatly simplified and cost effective compared to traditional agreements involving highly-paid lawyers. We discuss related work in Section V and Section VI presents concluding remarks.

II. Problem Formulation

We propose the use of interface automata for the formal specification of smart contracts. Interface automata build on the finite state machine model of computation to capture the states of communication, as well as relationships between input, output, and internal actions. An interface automaton was defined by Alfaro et al. as the tuple $P = \langle V_P, V_P^{init}, \mathcal{A}_P^I, \mathcal{A}_P^O, \mathcal{A}_P^H, \mathcal{T}_P \rangle$ [6] consisting of the following elements:

- V_P is the set of states,
- V_P^{init} is the set of initial states. V_P^{init} is required to contain at most one state. If $V_P = \emptyset$, then P is called empty.
- $\mathcal{A}_P^I, \mathcal{A}_P^O$, and $\mathcal{A}_P^H$ are mutually disjoint sets of input, output, and internal actions. $\mathcal{A}_P = \mathcal{A}_P^I \cup \mathcal{A}_P^O \cup \mathcal{A}_P^H$ denotes the set of all actions.
- $\mathcal{T}_P \subseteq V_P \times \mathcal{A}_P \times V_P$ is a set of steps. ■

We propose the use of interface automata to capture end user requirements, as well as the loyalty points issued or exchanged. We retain the *optimistic* approach to composition and the *alternating* approach to design refinement as defined in [6]. The optimistic approach in the context of loyalty points means that two automata expressing loyalty points resources and user requirements are compatible if there exists some environment in which they can work together. The alternating design refinement approach means that one interface refines another if it has weaker input assumptions and stronger output guarantees.

The problem formulation can be stated as follows. The proposed method captures the *(1)* loyalty points vendor resources, the *(2)* bids from all loyalty points consumers, and *(3)* loyalty points constraints, each modeled as interface automata using the concept of loyalty points as defined in Section II-A. The problem is then to find a composition of interface automata that can operate together using the *optimistic* approach to composition and the *alternating* approach to design refinement. If such an agreement exists, it is then recorded in the blockchain as a smart contract. The smart contract is unambiguous and forces each party to the contract to honor the agreement. If any party violates the contract, it can be detected automatically using the help of computers. This simplifies the agreement process and makes it cost effective for the management of the loyalty points marketplace.

A. Loyalty Points

Loyalty points are at the heart of the loyalty points marketplace. We formalize loyalty points as the tuple $L = \langle I, T, V \rangle$ consisting of the following elements:

- $I : L \rightarrow \mathcal{N}$ is a labeling function that assigns a natural number to the loyalty point L specifying the issuer of the loyalty point. Loyalty points issuers typically refer to companies looking to reward their customers. They can issue loyalty points out of thin air and assign monetary value to it themselves.
- $T : L \rightarrow \mathcal{N}$ is a labeling function that assigns a natural number to the loyalty point L specifying the type of the loyalty point. Loyalty points of the same type are issued by the same issuer and have the same value. For example, points on a Starbucks gift card are loyalty points of the same type.
- $V : L \rightarrow \mathcal{R}^+$ is a labeling function that assigns a real number to the loyalty point L specifying the value of the loyalty point. We capture value using the concept of *metacoins*. Each loyalty point has a corresponding metacoin value that may or may not correspond to an actual currency value (i.e. USD).

B. Loyalty Points Marketplace

Loyalty points are exchanged at the loyalty points marketplace. The loyalty points marketplace takes advantage of the decentralized, distributed and immutable platform provided by blockchain. We model loyalty points transactions using a two-party model.

a) Loyalty points vendor: The loyalty points vendor offers a set of loyalty points L^* for trade. We introduce the constraint that all loyalty points in the set are of the same type and issues by the same vendor $\forall l_i \in L^*, \forall l_j \in L^* : I_{l_i} = I_{l_j}, T_{l_i} = T_{l_j}$.

b) Loyalty points consumer: The loyalty point consumer bids for the set of loyalty points offered by the loyalty points vendor using a set of loyalty points L^{**}. Loyalty points offered for trade by the loyalty points consumer may be from different issuers and of different types.

c) Bidding process: The loyalty points vendor chooses a winning offer from the incoming bids. Since all loyalty points have a corresponding metacoin value, the loyalty points vendor can always map all incoming bids to a metacoin value. However, the selection algorithm may be arbitrary. For example, the loyalty points vendor may accept a lower bid if it means staying in his favorite hotel during his next travel rather than a higher value bid offering a free flight to someplace he does not plan on visiting soon.

d) Loyalty points transaction: Once a bid is selected, the exchange of loyalty points is recorded in blockchain as a transaction.

C. Loyalty Points Exchange Constraints

Loyalty points issuers wish to retain some level of control over loyalty points offered for exchange in the loyalty points marketplace. For example, a company may offer its loyal customers plenty of loyalty points with the hopes of retaining them. However, if there are too many loyalty points floating around in the marketplace, that might devalue the loyalty points offered by the issuer.

For example, if a customer has 10,000 loyalty points of the same type as a gift card for coffee, that customer may not mind to trade 1,000 of those loyalty points for a free hotel room stay valued at only 200 metacoins. This leads to devalue coffee gift card loyalty points, i.e. the metacoin value stated does not reflect the actual exchange rate between loyalty points of different types.

To counter this effect, loyalty points issuers can formalize constraints on the exchange of their loyalty points as smart contracts modeled as interface automata. Issuers can specify minimum exchange rates between loyalty points of certain types, they can prohibit exchange of loyalty points to other types of loyalty points, or simply tie the minimum exchange rate of their loyalty points to the metacoin value.

III. Loyalty Points Marketplace Case Study

Loyalty programs have been designed to encourage new or existing customers use of a business new or existing product and/or service in exchange for a reward (e.g. free amenities or reward points) that can be used at specific retailers or merchants. In a loyalty program, four entities work together to bring offerings to customers: issuers, business partners, retailers, and brokers. Issuers consist of business chains (e.g. hotel or retail chains) that provide the services or products to the customer; business partners include entities (e.g. hotels and airlines) that offer points to customers based on agreements and for their own benefit; retailers (e.g. coffee shops) consists of businesses that accept loyalty points in exchange for products that customers use; and brokers include entities (e.g. a blockchain service) that define the deals and transactions between issuers, business partners and customers. In this loyalty points marketplace case study, we present a decentralized loyalty program that enables different entities to interact through a smart exchange on the blockchain. The value of incorporating a loyalty marketplace platform is that it allows for exchange transactions across networks in different industries via a smart exchange, it captures user loyalty redemption and/or acquisition history for personalized services and marketing analysis, it provides a secure provenance of transaction history, it allows each blockchain owner to maintain full autonomy of business partners, and it offers a configurable marketplace for business-to-business defined and/or dynamic exchange rates.

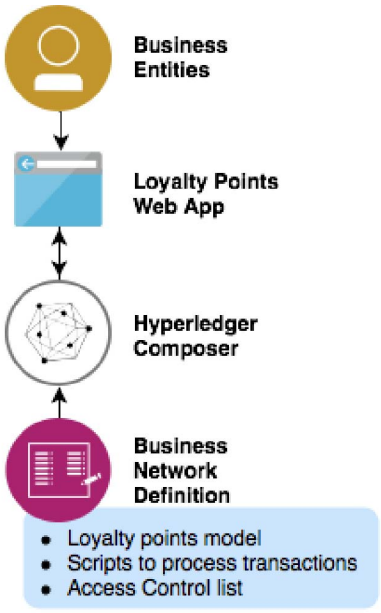

Fig. 1. Platform Architecture

Different types of market place models exist to offer better exchange rates, exchange conditions, exchange upon party agreements, and more. Four of these type of models include broker-controlled point exchange, B2B-controlled market place, business-controlled, and peer-to-peer point exchange market places. In the broker-controlled point exchange, the fixed exchange point rates are defined by the consortium and the system dynamically adapts based on the market value. The B2B-controlled market place offers better exchange rates to customers with preferred partners allowing for more favorable exchange rates as well as dynamic adjustment conversions to drive better incentives. The business partner independently offers better rates than the broker and partners agree upon. In business-controlled market places, businesses are the ones that define the accepted exchange conditions for their points through smart contract rules. And peer-to-peer market places exchange upon party agreements. Customers can trade among themselves with no regulation. In the proposed marketplace, the *Hyperledger Composer* [3] is used to model business networks, test these networks, and expose them via APIs. Hyperledger Composer is a framework for building Blockchain business networks that allows users to model their business network while preserving existing assets and transactions. A user can define the transactions that interact with various assets. A prototype of this loyalty points market place case has been implemented as a Web application and tested with various numbers of issuers, business partners, customers, loyalty points, and transactions.

In the proposed loyalty points marketplace, the system allows registration of various participants including business partners, retailers, and customers. Loyalty points can then be

issued to a loyalty point owner. Auctions can be created or closed, while bid transactions can be created. Throughout the process of registering participants and creating/closing auctions and/or bid requests, the system allows users to list all participants, all loyalty point types, and all auctions and transactions. Figure 1 shows the platform architecture used for the loyalty points web app developed for this case study.

In the proposed loyalty points marketplace, we define one business partner (e.g. a hotel chain), four customers (e.g. a family on vacation), four issuers (e.g. two coffee shops, a movie theater, or a museum), and one retailer (e.g. a coffee shop). After registering all participants, one or more loyalty points (e.g. movie award, adventurer pass, entertainment pass) are assigned to each of the customers. The loyalty point amount and type are recorded, along with the issuer whose loyalty type belongs to. Upon assigning loyalty point types, auctions can be created in which an existing loyalty point type assigned to a customer can be exchanged for a different loyalty point type. This auction phase allows a business entity to select the loyalty point type and the minimum amount that will be accepted by this entity in exchange. For example, the adventurer pass previously assigned to one of the four customers can be exchanged for a movie award at a specified minimum amount. Once the auctions have been created, bid transactions can be generated and sent to one of the auctions. Loyalty points can be selected along with the available loyalty point listings created during the auction. Auctions can be closed and details about each auction, including points for exchange and the amount, the seller and minimum amount, description of the auction and details can be inspected.

IV. Applying Smart Contracts to the Loyalty Points Marketplace

This section describes how the interface automata model of computation formalized in Section II can be used for the modeling of loyalty points transactions. We demonstrate how the interface automata provides a way to check the compatibility of requirements and resources provided.

The proposed method captures the *(1)* loyalty points vendor resources, the *(2)* bids from all loyalty points consumers, and *(3)* loyalty points constraints, each modeled as interface automata using the concept of loyalty points as defined in Section II. The problem is then to find a composition of interface automata that can operate together using the *optimistic* approach to composition and the *alternating* approach to design refinement.

A. Formal Modeling of the Loyalty Points Vendor

The loyalty points vendor is a role that several parties in the loyalty points marketplace may take on.

e) Issuers: Issuers play the role of the loyalty points vendor when they issue loyalty points to preferred customers. Loyalty points are issued to a specific loyalty points consumer. The issuer typically does not expect any compensation in return for issuing the loyalty points.

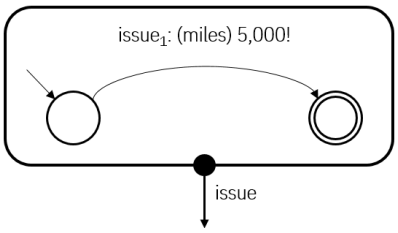

Fig. 2. Interface automaton model for the Issuer

f) Business partners: Business partners are entities willing to offer points to their customers for their own benefit. For example, certain hotels and airlines may form a parnership where they can exchange hotel points to and from airline miles. Business partners play the role of the loyalty points vendor when they offer loyalty points up for trade with their business partners.

g) Retailers: Retailers are entities and business associates that are not owned by the issuers, but they are willing to accept loyalty points in exchange for products. Retailers typically do not take on the role of a loyalty points vendor.

h) Customers: Customers are users of the facilities of issuers, business partners and retailers. They take on the role of a loyalty points vendor when they offer up their loyalty points for trade with other customers.

i) Broker: The broker is the service that runs in the blockchain. It defines deals between issuers, business partners and customers. The broker does not take on the role of a loyalty points vendor.

Fig. 2 shows the model for the Issuer in the loyalty points case study. The Issuer issues 5,000 loyalty points of type `miles` representing free airline miles to a loyalty points consumer. The loyalty points tuple L defined in Section II-A consists of $\langle \text{Issuer}, \texttt{miles}, 5,000 \rangle$. We formalize this by assigning the tuple L to the set of interface automata actions $\mathcal{A}_P$ defined in Section II. This concludes the formal modeling of the Issuer. ■

Metacoin values are assigned to the loyalty token type. For simplicity, we assume that each loyalty point type has a metacoin value of 1 in this example, giving the loyalty point package issued a total value of 5,000 metacoins.

B. Formal Modeling of the Loyalty Points Consumers

The loyalty points consumer is the opposite role to the loyalty points vendor that parties in the loyalty points marketplace may take on.

j) Issuers: Issuers do not take on the role of the loyalty points consumer.

k) Business partners: Business partners play the role of the loyalty points consumer when they accept loyalty points from their business partners.

l) Retailers: Retailers accept loyalty points in exchange for products. Thus, they take on the role of loyalty points consumer.

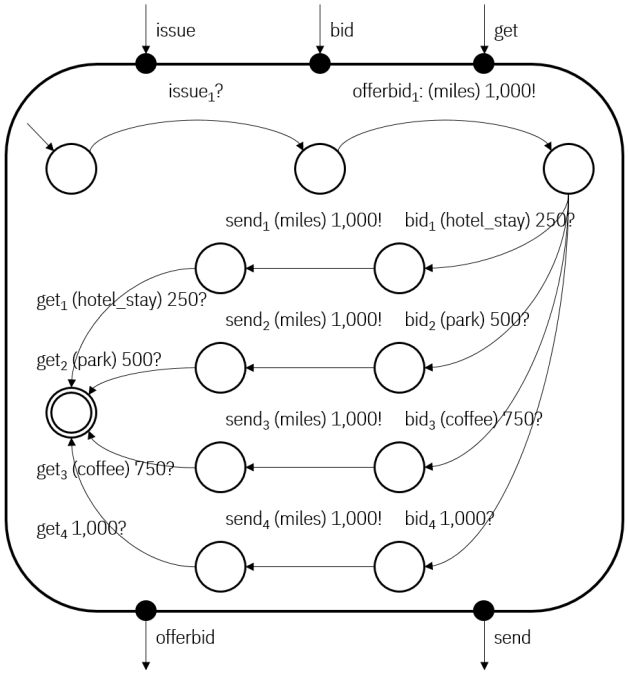

Fig. 3. Interface automaton model for Customer A offering loyalty points for trade

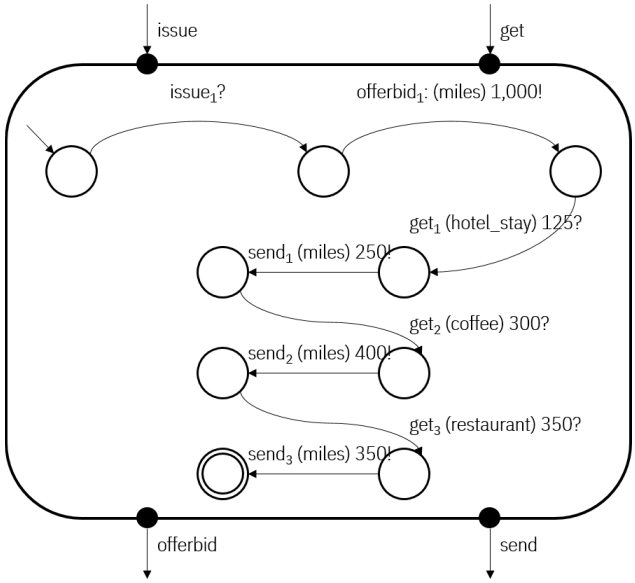

Fig. 4. Interface automaton model for the loyalty points exchange transaction by Customer A

m) Customers: Customers are users of the facilities of issuers, business partners and retailers. They take on the role of a loyalty points consumer when they accept loyalty points from issuers/business partners, or when they bid for loyalty points to trade with other customers.

n) Broker: The broker does not take on the role of a loyalty points consumer, as it is only interested in managing the exchange of loyalty points between other parties.

For this example, we model a customer that receives free airline miles from an issuer, playing the role of a loyalty points consumer. This customer then wants to trade some of his loyalty points and offers his airline miles up for bid, now playing the role of a loyalty points vendor. The customer would prefer to trade his airline miles for a free hotel stay. He wants this so bad he would be willing to accept a hotel stay worth 250 metacoins. He would also accept loyalty points for a theme park visit valued at 500 metacoins, or a gift card to a coffee place worth 750 metacoins. Finally, he would also accept loyalty points of undefined type with a minimum monetary value of 1 metacoins.

Fig. 3 shows the interface automaton model for the customer. The automaton captures the customer receiving loyalty points from the issuer. The customer then offers up 1,000 airline miles for trade. The minimum metacoin values for all types of loyalty points are captured in the model. the interface automaton shown in Fig. 3 is a smart contract on its own and is stored on the blockchain. It represents the types of deals that the customer expects and would deem acceptable. The minimum metacoin values are not revealed to potential bidders. This concludes the formal modeling of the Consumer. ■

C. Formal Modeling of the Transaction Constraints

Transaction constraints are imposed by issuers and/or business partners that play the role of loyalty points vendor. The purpose of such constraints is to protect the interests of the issuer after loyalty points are issued and prevent a devaluation of their loyalty points in the marketplace.

For this example, we consider a hotel that issues loyalty points. The hotel also specifies a minimum exchange rate of 0.5 metacoins to prevent the devaluation of free hotel stay loyalty points. We omit the interface automaton model for this constraint due to its simplicity, but more complex constraints can also be specified this way, by each type of loyalty point. An issuer might also prohibit the exchange of its loyalty points to loyalty points from their competitiors, or offer discounted loyalty point exchange rates to its business partners.

D. Composing Interface Automata into Smart Contracts

Consider customer B that bids on the 1,000 airline miles offered by customer A as shown in Fig. 3. Customer B has 125 loyalty points of type `hotel_stay`. In addition, he also has 300 loyalty points of type `coffee`. Finally, he has a restaurant gift card with 500 loyalty points on it. The bid can be captured as an interface automaton similar to the one shown in Fig. 3. We omit this model to preserve space.

The broker is in charge of matching bids to offers. The formal specification of the broker is out of scope for this paper.

In this subsection, we describe a possible implementation for the bid matching logic.

Customer A is willing to accept a minimum loyalty point exchange rate of 1:4, since he would be comfortable trading 250 loyalty points of type `hotel_stay` to 1,000 loyalty points of his own with type `miles`. There are two limitations, however. Customer B only has 125 loyalty points of type `hotel_stay`. Moreover, the hotel has imposed a minimum exchange rate of 0.5. As a result, customer A will have to use 250 of its `miles` loyalty points to trade for 125 `hotel_stay` loyalty points.

Customer A is further comfortable with an exchange rate of 3:4 to trade `miles` loyalty points for `coffee` loyalty points. Customer B has 300 `coffee` loyalty points avaliable for trade, for which customer A will trade 400 `miles` loyalty points. Customer A then has 350 `miles` loyalty points remaining, which he will trade to loyalty points of type `restaurant` with an exchange ratio of 1:1.

Once the bid matching process is complete, the transaction is captured as interface automata models on both customers' side and recorded in the blockchain. Fig. 4 shows the smart contract representation of the transaction corresponding to the offer by customer A shown in Fig. 3. ■

Note that each transaction recorded on the blockchain through the use of interface automata smart contracts may involve multiple exchanges between the parties. For example, Fig. 4 combines the transaction of receiving `miles` loyalty points issued with offering part of those loyalty points for trade, and finally trading them to loyalty points of 3 different types. The less frequently transaction are recorded, the more complex the automata get and thus the verification complexity also increases. On the other hand, recording each loyalty point exchange and/or bid as a transaction greatly simplifies the formal models, but leads to more transactions on the blockchain and thus poorer scalability.

E. Formal Verification of Smart Contracts modeled as Interface Automata

In the proposed method, model checking techniques are used *(1)* to find a possible matching between offers and bids that lead to an exchange of loyalty points that satisfy all constraints by all participants. We refer to this process as the *bid matching process*. Process *(2)* aims to verify that the transactions recorded in the blockchain are compatible with the bids, offers, and loyalty points resources are available to perform the requested transactions. We refer to this process as the *transaction verification process*.

The output of the interface automata composition described in Section IV-D may result in interface automata models of the following: *(a)* models for issuers issuing loyalty points, *(b)* models representing the exchange of loyalty points between business partners, *(c)* models capturing retailers accepting loyalty points in exchange for goods, *(d)* models capturing customers either accepting loyalty points from issuers, offering loyalty points up for trade, or bidding on loyalty points using their own loyalty points of different types.

Each model is captured as an interface automaton, with formally defined semantics and input/output actions. Thus, the input to both the bid matching process as well as the transaction verification process are a set of interface automata models that compose together and interact with each other through the set of actions $\mathcal{A}_P$ as defined in Section II.

o) Bid matching process: The bid matching process is concerned with finding a bid for loyalty points offered up for trade such that minimum metacoin values are met, and constraints imposed by the loyalty points issuers are satisfied. Once a match is found, the transaction may proceed and loyalty points are exchanged between market participants.

In the context of the formal interface automata models, the bid matching process can be formalized as reachability analysis over all the set of interface automata participating in the potential transaction. Since we use the *optimistic* approach to composition, we want to know whether an assignment exists such that the transaction may commence.

The bid matching can be formalized as reachability analysis as follows: does an execution trace exist over the set of automata such that each automaton reaches its end state?

The bid matching process can be implemented in many ways. It does not necessitate the use of formal methods. Solutions such as an online marketplace – i.e. eBay – or solutions that utilize Monte Carlo simulations to find all constraints are all perfectly acceptable. We prefer the reachability analysis because it simplifies step 2, the transaction verification process.

p) Transaction verification process: The transaction verification process aims to verify that a transaction defined over the set of interface automata participating in the transaction – such as the transaction shown in Fig. 4 – is *compatible* with the offers, bids – such as the automaton corresponding to the offer shown in Fig. 3 – and constraints imposed by the loyalty points issuers.

This is a crucial part of the proposed solution. The loyalty points marketplace is of no use to participants if the smart contracts are not enforced. We want to automate the transaction verification process to its fullest extent to reduce the cost of contractual disputes and to speed up transactions.

q) Implementation: We utilize the NuSMV model checker [8] for the formal verification of the interface automata models. Other model checkers such as the Symbolic Analysis Laboratory (SAL) [9] by SRI International are also suitable to implement the model checking methods described in this paper.

The process itself is fairly straightforward. We translate the interface automata models representing the issuers, business partners, retailers, and customers to the textual syntax used by the model checkers, and ensure that the events sent and received by the automata are properly connected, forming a network of automata models.

NuSMV does handle integer variables so constraints on numeric values can be enforced. Verification time for this example is less than a second. Scalability can become a problem however. The verification benefits if metacoin values

```
MODULE customer(issue, bid_hotel_stay_250,
    bid_park_500, bid_coffee_750,
    bid_generic_1000, send_hotel_stay_250,
    send_park_500, send_coffee_750,
    send_generic_1000)
VAR
  state: {start, issue1, offerbid1, bid1, bid2, bid3,
    bid4, send1, send2, send3, send4, end};
  offerbid_miles_1000: boolean;
  send_miles_1000: boolean;
  deal: boolean;
ASSIGN
  init(state) := start;
  next(state) :=
  case
    (state = start & issue): issue1;
    (state = issue1): offerbid1;
    (state = offerbid1 & bid_hotel_stay_250): bid1;
    (state = offerbid1 & bid_park_500): bid2;
    (state = offerbid1 & bid_coffee_750): bid3;
    (state = offerbid1 & bid_generic_1000): bid4;
    (state = bid1): send1;
    (state = bid2): send2;
    (state = bid3): send3;
    (state = bid4): send4;
    (state = send1 & send_hotel_stay_250): end;
    (state = send1 & send_park_500): end;
    (state = send1 & send_coffee_750): end;
    (state = send1 & send_generic_1000): end;
    TRUE: state;
  esac;
  init(offerbid_miles_1000) := FALSE;
  next(offerbid_miles_1000) :=
  case
    (state = issue1): TRUE;
    TRUE: FALSE;
  esac;
  init(send_miles_1000) := FALSE;
  next(send_miles_1000) :=
  case
    (state = bid1): TRUE;
    (state = bid2): TRUE;
    (state = bid3): TRUE;
    (state = bid4): TRUE;
    TRUE: FALSE;
  esac;
  init(deal) := FALSE;
  next(deal) :=
  case
    (state = end): TRUE;
    TRUE: FALSE;
  esac;
```

Fig. 5. NuSMV finite state machine model for the customer shown in Figure 3

are broken up to groups of metacoins. So, the value 500 may be represented only as 5. For best scalability, boolean values can be used but then the enforcement of token values falls on the blockchain. Again, it comes down to a tradeoff between verification and blockchain scalability. The size of the automata models must remain limited for better scalability, but also for easier understandability by the users.

We introduce the boolean variable `deal` to denote the state when a deal is reached for the consumer. We have similar constructs for the bidder. The model for the bidder and issuer is omitted to save space. We use a set of *Computational Tree Logic (CTL)* [10] formulas to show that a deal is possible between the customer and the bidder.

- `(AG EF issuer1.state = end)`
- `(AG EF bidder1.state = end)`
- `(AG EF bidder1.deal)`
- `(AG EF customer1.state = end)`
- `(AG EF customer1.deal)`

For example, the formula `AG EF customer1.deal` shows that on all paths in the CTL branching time logic, eventually there is such a future state that the customer1.deal variable evaluates to true, meaning the customer has successfully traded airline miles worth 1000 metacoins for a hotel stay worth 250 metacoins.

V. Related Work

Smart contracts help facilitate and enforce agreements between parties. However, it is an ongoing task to ensure that complex contractual clauses of smart contracts are enforced and any violations of the agreement are detected and dealt with appropriately. Several studies have been performed to identify current topics and challenges of smart contracts [11]. In this study, several security, codifying, privacy, performance issues, and other related works were identified. In particular, security issue related works such as criminal activities, mishandled exceptions, reentrancy and untrustworthy data feeds, and others were examined.

A method for formal requirement enforcement on smart contracts using linear dynamic logic was presented by Sato et al [12]. The authors describe a tool chain where domain-specific languages are translated to UML statecharts. The tool integrates into Hyperledger [3] to enforce constraints. In contrast, the method presented in this paper focuses on the formal verification of smart contracts.

An approach to generate smart contracts was presented by Choudhury et al [13]. The authors use a context-free grammar to parse natural language and create formal smart contracts. In contrast, this paper utilizes the interface automata model of computation. The formal models themselves are the smart contracts, thus avoiding potential ambiguity from using natural languages.

Some works have focused on resolving concerns within the smart contract life cycle development. Liao et al. [14] propose a platform that provides automated integration services in order to reduce the complexity and difficulty when designing and testing smart contracts. Their proposed platform is demonstrated by explaining how it helps users develop and test a loyalty points exchange smart contract.

A different study [15] explores the research challenges and issues with regards to the process of validating and verifying smart contracts. They identified several questions that must be answered in order to ensure that a smart contract program executes correctly. A formal representation of a smart contract and its effects are needed.

Several modeling approaches have been proposed to verify smart contract behavior. In one approach [16], a model checking language is used to model blockchain behavior. An analysis on the safety is used to analyze the safety of the

register smart contract execution by introducing a model that steals a user's identity.

Frameworks have also been introduced to validate and check the correctness of smart contracts. The Zeus framework [17] consists of a tool that builds policies, translates source code, and verifies smart contracts. The smart contract along with a set of policy assertions is converted to LLVM bit code. Zeus verifier determines if there are any policy violations. Their evaluations indicated that a large percentage of smart contracts are vulnerable to bugs.

Interface automata has been used to verify several component-based systems, and numerous applications and extensions of this formal model exist. Kim et al [18] extend interface automata with asynchronous I/O and derive new composition operators. Modal interface automata were introduced by Lüttgen et al [19]. A method to formalize hardware/software interfaces in the Windows Driver Framework was described in [20]. The authors believe that the use of the interface automata model of computation for the verification of Blockchain smart contracts, however, is a novel idea that has not been previously explored.

VI. Concluding Remarks

This paper presented a system and method for the formal verification of smart contracts for a loyalty points case study. The interface automata model of computation [6] is used to capture loyalty points transactions.

We retain the *optimistic* approach to composition and the *alternating* approach to design refinement. The optimistic approach in the context of loyalty points means that two automata expressing loyalty points resources and user requirements are compatible if there exists some environment in which they can work together. The alternating design refinement approach means that one interface refines another if it has weaker input assumptions and stronger output guarantees.

A formal approach is necessary to unambigously capture *smart contracts* between participants of the loyalty points marketplace. By formally modeling the end user requirements and resources provided by the loyalty points vendor, one can formally verify whether the requirements and provided resources are *compatible*. Violations between the smart contract and actual implementation can then be detected and enforced by either party to the agreement.

We presented an approach that utilizes model checking techniques to verify whether automata models between customers and business partners in a loyalty points marketplace are compatible. We showed the feasibility of the approach to capture business agreements between formal smart contracts through a simple bidding process. The case study used for the loyalty points marketplace was implemented on Hyperledger.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] E. Foundation, "Ethereum," 2017. [Online]. Available: https://www.ethereum.org/

[3] L. Foundation, "The hyperledger project," 2017. [Online]. Available: https://www.hyperledger.org

[4] N. Alchemy, "A short history of smart contract hacks on ethereum: A.k.a. why you need a smart contract security audit," 2019. [Online]. Available: https://medium.com/new-alchemy/a-short-history-of-smart-contract-hacks-on-ethereum-1a30020b5fd

[5] D. Labs, "Cryptokitties: Collect and breed digital cats," 2016. [Online]. Available: https://www.cryptokitties.co/

[6] L. de Alfaro and T. A. Henzinger, "Interface Automata," in *Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. ESEC/FSE-9, 2001, pp. 109–120.

[7] L. Foundation, "The hyperledger project," 2017. [Online]. Available: https://github.com/hyperledger/composer

[8] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking," in *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, ser. LNCS, vol. 2404. Copenhagen, Denmark: Springer, July 2002.

[9] N. Shankar, "Symbolic analysis of transition systems," in *Abstract State Machines: Theory and Applications (ASM 2000)*, ser. Lecture Notes in Computer Science, Y. Gurevich, P. W. Kutter, M. Odersky, and L. Thiele, Eds., no. 1912. Monte Verità, Switzerland: Springer-Verlag, mar 2000, pp. 287–302. [Online]. Available: http://www.csl.sri.com/papers/asm2000/

[10] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching-time temporal logic," in *Logic of Programs, Workshop*, 1982, pp. 52–71.

[11] M. Alharby and A. van Moorsel, "Blockchain-based smart contracts: A systematic mapping study," *arXiv preprint arXiv:1710.06372*, 2017.

[12] N. Sato, T. Tateishi, and S. Amano, "Formal requirement enforcement on smart contracts based on linear dynamic logic," in *IEEE Blockchain*, 2018, pp. 945–954.

[13] O. Choudhury, N. Rudolph, I. Sylla, N. Fairoza, and A. Das, "Auto-generation of smart contracts from domain-specific ontologies and semantic rules," in *IEEE Blockchain*, 2018, pp. 963–970.

[14] C.-F. Liao, C.-J. Cheng, K. Chen, C.-H. Lai, T. Chiu, and C. Wu-Lee, "Toward a service platform for developing smart contracts on blockchain in bdd and tdd styles," in *2017 IEEE 10th Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2017, pp. 133–140.

[15] D. Magazzeni, P. McBurney, and W. Nash, "Validation and verification of smart contracts: A research agenda," *Computer*, vol. 50, no. 9, pp. 50–57, 2017.

[16] T. Abdellatif and K.-L. Brousmiche, "Formal verification of smart contracts based on users and blockchain behaviors models," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE, 2018, pp. 1–5.

[17] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "Zeus: Analyzing safety of smart contracts," in *25th Annual Network and Distributed System Security Symposium (NDSS?18)*, 2018.

[18] K. Larsen, U. Nyman, and A. Wasowski, "Interface input/output automata," in *FM 2006: Formal Methods*, 08 2006, pp. 82–97.

[19] W. V. Gerald Lüttgen, "Modal interface automata," in *Theoretical Computer Science*, 2012, pp. 265–279.

[20] J. Li, F. Xie, T. Ball, V. Levin, and C. McGarvey, "Formalizing hardware/software interface specifications," in *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), Lawrence, KS, USA, November 6-10, 2011*, 2011, pp. 143–152.