UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MARCOS BARROS DE MEDEIROS FILHO

CONTRATO INTELIGENTE DE TRÊS PONTAS: PAGAMENTOS SEGUROS

CAMPINA GRANDE - PB

2022

MARCOS BARROS DE MEDEIROS FILHO

CONTRATO INTELIGENTE DE TRÊS PONTAS: PAGAMENTOS SEGUROS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador : José Antão Beltrão Moura

CAMPINA GRANDE - PB

2022

MARCOS BARROS DE MEDEIROS FILHO

CONTRATO INTELIGENTE DE TRÊS PONTAS: PAGAMENTOS SEGUROS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

José Antão Beltrão Moura

Orientador – UASC/CEEI/UFCG

Marcus Salerno de Aquino

Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 02 de Setembro de 2022.

CAMPINA GRANDE - PB

RESUMO

Contratos inteligentes são programas executados de forma descentralizada em uma rede de computadores que tem o poder de descrever um fluxo de eventos e consequências para cada ação. Um dos desafios ainda existentes é como modelar eventos do mundo real que não podem ser obtidos por meios digitais ou oracles descentralizados. O presente trabalho desenvolve uma aplicação modelo, utilizando um contrato inteligente na rede Ethereum, com caso de uso real, para estudar a modelagem de uma interface que permite que informações não digitalizadas possam ser inseridas e validadas por agentes externos à blockchain, além de explorar as técnicas de desenvolvimento de contratos inteligentes e suas limitações.

Contrato inteligente de três pontas: Pagamentos Seguros

Marcos Barros de Medeiros Filho
Universidade Federal de Campina Grande
Campina Grande, Brasil.
Graduando em Ciência da Computação
marcosbarrosmf@gmail.com

José Antão B. Moura
Universidade Federal de Campina Grande
Campina Grande, Brasil
Orientador
antao@computacao.ufcg.edu.br

Tiago L.P. Clementino
Universidade Federal de Campina Grande
Campina Grande, Brasil
Co-Orientador
tiagolucas@copin.ufcg.edu.br

RESUMO

Contratos inteligentes são programas executados de forma descentralizada em uma rede de computadores que tem o poder de descrever um fluxo de eventos e consequências para cada ação. Um dos desafios ainda existentes é como modelar eventos do mundo real que não podem ser obtidos por meios digitais ou oracles descentralizados. O presente trabalho desenvolve uma aplicação modelo, utilizando um contrato inteligente na rede Ethereum, com caso de uso real, para estudar a modelagem de uma interface que permite que informações não digitalizadas possam ser inseridas e validadas por agentes externos à blockchain, além de explorar as técnicas de desenvolvimento de contratos inteligentes e suas limitações.

Palavras-chave

Smart Contracts, Contratos Inteligentes, Blockchain, Ethereum, Solidity, Truffle, Ganache, Pagamento Seguro, Redes Descentralizadas, Event Oracle.

Repositórios

https://github.com/conditional-token/SafePayment

1. INTRODUÇÃO

Em 2008, Satoshi Nakamoto lançou a ideia da criptomoeda Bitcoin, uma moeda digital[1], utilizando-se de uma estrutura de dados que foi chamada de Blockchain, um banco de dados distribuído que utiliza de criptografia e algoritmos de consenso para manter um histórico de transações praticamente inalterável.

A ideia do Bitcoin se popularizou e seu valor de mercado chegou, em abril de 2022, a cerca de 1 trilhão de dólares [2]. Outros projetos inspirados pelo Bitcoin também se popularizaram, com focos mais específicos, como garantia de anonimato na rede Monero, e outros com a ideia de estender ainda mais as funcionalidades de uma rede descentralizada com a criação de contratos inteligentes, como a rede Ethereum.

Contrato inteligente é um termo inicialmente cunhado por Nick Szabo, em 1996. Foi descrito como um conjunto de promessas, especificado em formato digital, incluindo os protocolos pelos quais as partes envolvidas podem executar essas promessas [3].

Em 2014, inspirado nas idéias de Nick Szabo e na rede descentralizada Bitcoin, Vitalik Buterin lança a descrição de uma rede que expande as capacidades de uma blockchain com uma implementação de uma linguagem turing-completa, permitindo que qualquer pessoa escreva contratos inteligentes e aplicações descentralizadas com regras programáveis, formatos de transação e funções de transição [4].

As primeiras aplicações populares de contratos inteligentes com Ethereum foram na área de finanças descentralizadas, como a criação de moedas diferentes dentro da rede, e na área de jogos, permitindo que usuários pudessem ter posse e transacionar objetos digitais que tinham semântica dentro dos jogos.

Em 2022, uma revisão de literatura realizada por Tiago Lucas observou que um dos desafios para maior adoção de contratos inteligentes é a dificuldade de virtualização de valores e eventos do mundo real [5]. Dessa forma, o presente trabalho propõe endereçar essa dificuldade através do desenvolvimento de um contrato inteligente que utiliza de eventos externos a blockchain para ser executado, de forma a contribuir com o relato das dificuldades encontradas, limitações de tecnologias e uma proposta de interface para a virtualização de eventos do mundo real que podem ser validados por terceiros em um ambiente de computação descentralizada. De forma complementar, um outro trabalho de conclusão de curso, desenvolvido por Caio Sanches[6], se propôs a criar uma interface web que facilita a interação entre usuários comuns e um contrato inteligente, utilizando o contrato desenvolvido neste trabalho.

2. APLICAÇÃO MODELO

Marketplaces podem ser definidos como um shopping virtual, um site de e-commerce que reúne ofertas de produtos e serviços de diferentes vendedores [7]. A maior parte dos marketplaces implementam uma funcionalidade que facilita muito a questão da confiança entre o vendedor e comprador, e que aqui vamos chamar de pagamento seguro.

O pagamento seguro consiste no marketplace agindo como intermediário de um pagamento entre comprador e vendedor. O comprador paga ao marketplace, que por sua vez retém o pagamento, garantindo que o vendedor receberá o pagamento apenas caso entregue o produto ou serviço. Com a entrega do

produto confirmada, o pagamento é finalmente liberado para o vendedor (Figura 1).

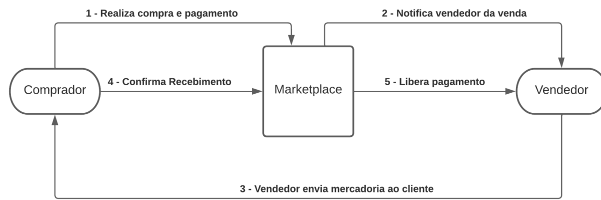

Figura 1. Diagrama do Pagamento Seguro

Apesar de o fluxo parecer simples, em caso de conflito entre o comprador e o vendedor, fica a cargo do marketplace levantar os fatos e tomar uma decisão quanto ao pagamento. O processo de resolução de disputa pode ser automatizado até certo ponto, para tomar uma decisão é preciso avaliar os argumentos de ambos os lados e utilizar do contrato realizado entre as partes, bem como de legislações locais, para decidir quem quebrou o contrato e deve ter prejuízo.

2.1 O problema dos dados não digitais

Trazer informações externas para a blockchain é um problema frequente na literatura especializada. Diversas aplicações financeiras descentralizadas precisam de informações externas à blockchain, como a cotação atual do câmbio entre duas moedas. Como acessar essa informação de forma descentralizada e com confiança na informação recebida? Dado que uma informação errada pode causar prejuízos enormes para os diversos atores envolvidos.

Com esse problema em mente, foi criado o conceito de Oracles. Oracles são entidades externas a blockchain a quem é confiado o papel de prover dados do mundo real de forma a cumprir as condições necessárias para execução dos contratos inteligentes [8]. Para tornar esse serviço mais seguro, algumas redes implementaram oracles descentralizados, grandes redes de nós que recuperam informações externas e entram em consenso para gerar uma resposta unificada. Dessa forma o risco de um ator malicioso prover informações falsas para a blockchain é reduzido.

Os oracles normalmente são utilizados para recuperar informações disponíveis em canais digitais, como as disponibilizadas por servidores na internet, mas não permitem averiguar eventos que ocorrem fora do mundo digital alcançável.

2.2 Requisitos de um Pagamento Seguro Descentralizado

Com o modelo implementado por marketplaces em vista, bem como os conceitos de descentralização de contratos e oracles, foram levantados os seguintes requisitos para o pagamento seguro descentralizado:

São compreendidas três entidades que participam do contrato, o emissor do pagamento, o destinatário e os validadores, uma entidade que irá validar as condições definidas pelas partes para que o pagamento ocorra. Na rede Ethereum, toda entidade, seja contrato, seja usuário, possui um endereço público que a identifica. Dessa forma, qualquer das entidades do nosso contrato pode ser um usuário humano ou uma máquina.

Dessa forma, foram decididos os seguintes requisitos:

- O pagamento seguro é uma ordem de pagamento entre duas partes onde uma condição precisa ser satisfeita para a liberação dos fundos.
- O papel de validador da transação deve ser livremente definido entre as entidades participantes e informado na criação da transação.
- O pagamento pode ser aprovado ou recusado. Em caso de aprovação, o destinatário consegue sacar o valor. Em caso de reprovação, o emissor do pagamento consegue recuperar seus fundos.
- Para melhorar a resiliência, um pagamento pode ter múltiplos validadores.

3. METODOLOGIA

Com o objetivo de atacar a dificuldade apresentada por Tiago Lucas em sua revisão literária, foi definido o desenvolvimento de uma aplicação, em formato de contrato inteligente, que tem como base de execução a virtualização de eventos do mundo real, como o julgamento de disputas. A execução do projeto foi feita em quatro etapas, compreensão do problema e definição dos requisitos, escolha de tecnologias a serem utilizadas e desenvolvimento.

3.1 Definição da aplicação modelo e requisitos do contrato inteligente

O problema do pagamento seguro foi definido pela dificuldade de se escrever de maneira sólida e invariável um algoritmo que resolva uma disputa entre duas partes que podem utilizar fatos e argumentos não verificáveis de forma digital, necessitando da intervenção de um terceiro com conhecimento suficiente para julgar o ocorrido.

A definição de quais seriam os requisitos funcionais se deu por meio de discussão entre os autores, em reuniões semanais, levando em consideração o ponto de vista do usuário e tornar o contrato genérico o suficiente para ser utilizado em qualquer tipo de pagamento, usando a premissa que uma entidade deve agir como intermediária e julgar as disputas. O período de discussão do modelo e fechamento do escopo do projeto foi de 3 semanas.

3.2 Escolha de tecnologias

A escolha das tecnologias se deu através da busca de ferramentas populares para criação de contratos inteligentes, levando em consideração requisitos não funcionais como possibilidade de criar testes e rodar a aplicação em um ambiente de desenvolvimento local.

3.3 Desenvolvimento

O desenvolvimento se deu completamente por um único desenvolvedor, com apoio do co-orientador na revisão de código. Foi utilizado uma metodologia scrum modificada, com reuniões semanais para discutir os avanços e dificuldades encontradas no desenvolvimento. O período de desenvolvimento compreendeu cerca de 4 semanas, que incluiu o aprendizado das tecnologias e busca por padrões de desenvolvimento de contratos inteligentes, buscando escrever um código limpo e bem testado.

3.4 Relato de desenvolvimento

Por fim, a última etapa é relativa à escrita do presente trabalho, que é um relato de desenvolvimento, compartilhando a experiência, bem como as limitações e desafios, do desenvolvimento de uma aplicação descentralizada que depende de dados não digitalizados.

4. SOLUÇÃO

4.1 Das tecnologias e suas limitações

4.1.1 Solidity

Um contrato inteligente na rede Ethereum é um código escrito em uma linguagem de programação que é interpretada pela Ethereum Virtual Machine, EVM, uma especificação de máquina virtual que executa os programas na rede Ethereum. Existem diversas implementações da EVM e linguagens de programação que compilam para um código que pode ser executado por ela, sendo Solidity a mais popular.

Um contrato pode ser descrito como um conjunto de estados e funções. Ao ser implantado na rede, o contrato vira um endereço público, uma entidade totalmente independente, controlada apenas pelo seu próprio código. A interação com o contrato se dá por transações que especificam a invocação de funções do contrato. É importante notar que, como um dos princípios mais importantes da blockchain está a imutabilidade, sendo assim, todas as interações com um contrato inteligente são irreversíveis e geram registros históricos na rede.

A descentralização faz com que a execução de programas complexos na rede Ethereum sejam extremamente custosos. O estado definido por um contrato precisa ser replicado entre os nós da rede e sua modificação precisa ser calculada por diversos nós com o fim de gerar um consenso para o estado da rede após a transação. Dessa forma, para implementar um contrato viável, é preciso ser cuidadoso quanto a forma de manipulação e armazenamento dos dados necessários para a execução de suas funcionalidades.

O presente trabalho utiliza a Solidity para implementar um contrato executável em Ethereum Virtual Machines.

4.1.2 Ganache

Uma das grandes complicações no desenvolvimento de software para redes descentralizadas é que as interações com a rede podem ser bem custosas. Ethereum, por exemplo, possui um sistema em que várias transações, ou chamadas de método, na rede são agrupadas, formando blocos que são processados em conjunto. Dessa forma, ações que modificam o estado do contrato precisam entrar em um bloco, ser executadas por diversos nós e por fim a rede chegar a um consenso sobre seu estado final. Por consequência, setup e execução de testes viram um processo extremamente lento. Para resolver esse problema temos o Ganache, uma implementação de rede Ethereum pessoal que pode ser utilizada para simular o comportamento da rede com apenas um nó, transações instantâneas e fácil setup de carteiras pré-carregadas com a moeda necessária para transacionar na rede.

4.1.3 Truffle

Como chamadas de funções que alteram o estado do contrato em Solidity são transações na rede, os testes de um contrato inteligente acabam sendo muito voltados a um teste de integração. É possível escrever testes em Solidity, mas também existe uma variedade de frameworks que permitem escrever testes completos em linguagens de programação diferentes. Truffle é um framework de desenvolvimento de contratos inteligentes que permite utilizar Javascript para realizar testes de contratos inteligentes usando frameworks já bem estabelecidos de testes orientados a comportamento. Por maior familiaridade e facilidade no desenvolvimento de testes usando Javascript, Truffle foi escolhido para dar suporte aos testes do contrato.

Por fim, Truffle também nos permite realizar a chamada RPC para implantação do contrato em ambientes locais, de teste e produção. Dessa forma, através de pequenas configurações e fornecendo uma carteira com saldo suficiente, podemos realizar o deploy do nosso contrato em qualquer rede.

4.2 Da modelagem dos dados

A EVM trabalha com unidades de dados de 256 bytes, isso significa que independente do espaço necessário, qualquer tipo de dado persistido no estado de um contrato irá utilizar múltiplos de 256 bytes.

O custo das chamadas de função na rede, também chamadas de transações, é calculado pelas operações realizadas, tanto em processamento, quanto em armazenamento. O armazenamento de dados acaba sendo uma das operações mais caras e determinísticas para o custo de operações que não envolvem grande utilização de recursos de processamento.

Por exemplo, a operação que salva um dado qualquer de 256 bytes na EVM é chamada de SSTORE e possui um custo fixo de 20 mil gas para cada chamada, podendo equivaler a alguns dólares para gravar uma única variável[9].

O processamento também pode se tornar custoso à medida que nosso programa começa a percorrer listas cada vez maiores para buscar informações, dessa forma, pode ser preferível, por exemplo, utilizar uma estrutura de dados auxiliar que permita uma busca mais rápida.

Dessa forma, a modelagem levou em consideração os dados mínimos que um contrato precisa para ser executado, tentando não onerar mais que o necessário os seus utilizadores, mas também prover formas de realizar buscas rápidas nos pagamentos e validadores.

Além de utilizar uma lista com os validadores, permitindo que aplicações externas consultem todos os possíveis validadores de um pagamento, também foi utilizado um, um mapping, tipo primitivo de Solidity equivalente a um Hashmap, para permitir verificação rápida . Algumas flags são utilizadas para realizar o controle do estado do pagamento (Figura 2).

Payment	
issuer	address
receiver	address
validators	[]address
isValidator	mapping(address => bool)
isValidated	bool
isApproved	bool
isPaid	bool
paymentValue	uint256
validationFee	uint256

Figura 2. Modelo de dados do pagamento seguro

Com a estrutura que define um pagamento pronta, ainda é necessário prover uma forma de referenciar um pagamento específico ao interagir com o contrato, e índices que facilitem encontrar quais pagamentos estão relacionados com uma carteira específica. Dessa forma, foram utilizados mappings para permitir a busca rápida através de índices que mapeiam endereços para o id dos pagamentos, bem como um mapping que mapeia ids para pagamentos concretos. Vale ressaltar que o identificador não está presente na estrutura do pagamento para reduzir custos no seu armazenamento. Por fim, também foi necessária uma variável para guardar o contador de pagamentos, facilitando a operação de encontrar um identificador livre para um novo pagamento (Figura 3).

ContractState	
payments	mapping(uint256 => Payment)
nextPaymentID	uint256
issuerIndex	mapping(address => []uint256)
receiverIndex	mapping(address => []uint256)
validatorIndex	mapping(address => []uint256)

Figura 3. Modelagem do estado do contrato

4.3 Dos atores e comportamentos

O programa prevê o uso de três atores com papéis e direitos distintos, sendo eles o emissor do pagamento, os possíveis validadores e o destinatário. A parte de algumas funções de visualização do estado do contrato e índices de busca, a lógica bruta ocorre em três momentos: criação do pagamento, validação e reivindicação do valor.

A etapa de criação de um pagamento é totalmente responsabilidade do emissor. Ele deve informar quem é o destinatário, possíveis validadores, o valor do pagamento e uma taxa que será paga ao validador no ato de validação de um pagamento. Note que o emissor do pagamento pode decidir sozinho quem é o validador, cabe ao destinatário conferir se está de acordo com os endereços apontados no contrato antes de realizar o serviço ou transferência de bens (Figura 4).

```
/// @notice is the payment event constructor.
/// @param paymentValue is the value that must be paid to payable to. It must be sent in the tx value.
/// @param validationFee is the fee for the validator. It must be sent in the tx value.
/// @param receiver the address that this payment is addressed to.
/// @param validators an array of addresses that can validate this payment.
function createPayment(
    uint256 paymentValue,
    uint256 validationFee,
    address receiver,
    address[] calldata validators
) public payable returns (uint256) {
    require(paymentValue + validationFee == msg.value, "Value must be equal paymentValue and validationFee");
    contractBalance += msg.value;

    uint256 paymentID = nextPaymentID;
    nextPaymentID++;

    Payment storage p = payments[paymentID];
    p.issuer = msg.sender;
    p.paymentValue = paymentValue;
    p.receiver = receiver;
    p.validationFee = validationFee;
    p.validators = validators;
    for(uint256 i=0; i < validators.length; i++) {
        p.isValidator[validators[i]] = true;
        // validator index
        validatorIndex[validators[i]].push(paymentID);
    }
    issuerIndex[msg.sender].push(paymentID);
    receiverIndex[receiver].push(paymentID);

    address[] memory parties = new address[](1);
    parties[0] = receiver;
    emit EventCreated(paymentID, msg.sender, parties, validators);

    return paymentID;
}
```

Figura 4. Função responsável pela criação do pagamento

A segunda etapa diz respeito a validação, nesse momento o valor está retido no contrato e só pode ser liberado, seja como reembolso para o emissor ou pagamento efetivo para o destinatário, após a validação de um dos validadores apontados. É possível aprovar ou rejeitar o pagamento. Em caso de rejeitar, o validador está afirmando que o pagamento não deve ser feito, e, por consequência, deve ser estornado para o emissor. Em caso de aceite, o destinatário pode fazer a reivindicação do pagamento. Ambas as funções de aprovação e rejeição possuem uma taxa, um valor inteiro que indica a força com que se está rejeitando ou aprovando algo, no caso do contrato de pagamento ele é ignorado e apenas utilizado com o fim de implementar a interface de evento validável genérico (Figura 5). Ao realizar a validação, o agente validador também recebe a taxa de validação definida na criação do pagamento como prêmio pelo serviço prestado.

```
/// @notice allows sender to reject an event. Should raise error if event was already validated.
/// For this contract, rates are ignored.
/// @param paymentID Identifier of the event.
/// @param rejectRate A rate that can be used by contracts to measure approval when needed.
function rejectEvent(uint256 paymentID, uint256 rejectRate) external override {
    Payment storage p = payments[paymentID];
    require(p.issuer != address(0), "payment id doesn't exist.");
    require(!p.isValidated, "payment was already validated");
    require(p.isValidator[msg.sender], "msg.sender is not a valid validator for the payment");

    p.isApproved = false;
    p.isValidated = true;
    _transfer(payable(msg.sender), p.validationFee);
    emit EventRejected(paymentID, msg.sender, rejectRate);
}

/// @notice allows sender to approve an event. Should raise error if event was already validated.
/// @param paymentID Identifier of the event.
/// @param approvalRate A rate that can be used by contracts to measure approval when needed.
function approveEvent(uint256 paymentID, uint256 approvalRate) external override {
    Payment storage p = payments[paymentID];
    require(p.issuer != address(0), "payment id doesn't exist.");
    require(!p.isValidated, "payment was already validated");
    require(p.isValidator[msg.sender], "msg.sender is not a valid validator for the payment");

    p.isApproved = true;
    p.isValidated = true;
    _transfer(payable(msg.sender), p.validationFee);
    emit EventApproved(paymentID, msg.sender, approvalRate);
}
```

Figura 5. Funções de validação

Por fim, há a etapa de reivindicação do pagamento. Essa etapa pode ser executada pelo emissor em caso de pagamento rejeitado, ou pelo destinatário em caso de pagamento aprovado (Figura 6). A decisão de requerer uma ação manual na reivindicação do pagamento aconteceu com o fim de não onerar o validador com a taxa cobrada pela rede na transferência de valores.

```
/// @notice claimPayment allows the payment target to withdraw the value from the contract after the payment was validated.
/// It also allows issuer to retrieve the money is payment was not approved.
/// @param eventID is the payment ID.
function claimPayment(uint256 eventID) external {
    Payment storage p = payments[eventID];
    require(p.id != 0, "payment id doesn't exist.");
    require(p.isValidated, "payment wasn't validated.");
    if (p.isApproved) {
        require(p.receiver == msg.sender,"msg.sender is not the receiver of the payment.");
    } else {
        require(p.issuer == msg.sender, "msg.sender is not the issuer of this payment.");
    }
    require(!p.isPaid, "payment was already made.");

    _transfer(payable(msg.sender), p.paymentValue);
    p.isPaid = true;
}
```

Figura 6. Função de reivindicação do pagamento

4.4 Da interface genérica de evento validável

Um dos objetivos do trabalho é buscar uma interface genérica para representar eventos não digitalizados que podem ser validados por alguma entidade, como a ocorrência de um serviço ou transferência de bens que são as condições para a liberação de um pagamento. Um evento verificável pode ser informado antes ou depois que ele aconteça. No caso de um contrato de pagamento, o evento é informado antes como requisito para o pagamento. Pensando em um registro de julgamento de um crime, o evento apresentado aconteceu antes de sua introdução na blockchain. Desta forma, um evento pode ser validado a qualquer momento.

Outro caso de uso de se pensar é em uma validação parcial, como no caso de licitações onde o pagamento é liberado à medida que a obra é concluída, o evento de construção pode não ser representado com um sim ou não, mas com um número entre 0 e 100 representando a porcentagem de conclusão daquela obra.

Com essas características em vista, foi desenvolvida uma interface, que tenta propor um modelo de comportamento para validação de eventos externos em contratos inteligentes. A interface busca ser o mais genérica possível, não fazendo juízo de como a estrutura de dados de um evento validável deve ser, requer apenas que seja passível de identificação por um ID do tipo uint256, de forma que as funções do contrato possam identificar em qual evento estão agindo.

O evento validável deve possuir uma regra para definir quem são seus validadores, ele deve permitir que o validador realize a aprovação ou rejeição do evento com a utilização de um grau de aprovação/rejeição opcional. O evento também deve implementar uma lógica que define seu estado de aprovação (Figura 7).

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

// ValidableEvent is an interface to represent a validatable event.
// An event can represent anything. It must have an unique ID which identifies it.
// The contract that implements should have it's own policy of event validation.
// The event might have a approvalRate and any logic can be associated with it, since approve and reject functions both accept a rate.
interface ValidableEvent {

  /// @notice Get the status of an event. Status is represented by two boleans, isApproved and isFinal.
  /// @param eventID Identifier of the event.
  /// @return isApproved bool representing if event was approved.
  /// @return isFinal bool representing if event was validated and isApproved is a final approval status.
  function getEventStatus(uint256 eventID) external view returns(bool isApproved, bool isFinal);

  /// @notice Get the approval and reject rates of the event.
  /// @param eventID Identifier of the event.
  /// @return approvalRate uint256 representing the rate of approval.
  /// @return rejectRate uint256 representing the rate of rejection.
  function getEventRates(uint256 eventID) external view returns(uint256 approvalRate, uint256 rejectRate);

  /// @notice Returns possible addresses that can validate the event with eventID.
  /// @param eventID Identifier of the event.
  /// @return address of validator.
  function getEventValidators(uint256 eventID) external view returns(address[] memory);

  /// @notice Returns a boolean if the address is a possible event validator.
  /// @param eventID Identifier of the event.
  /// @param validator Address of the possible validator.
  /// @return bool representing if validator can approve or reject the event with eventID.
  function isEventValidator(uint256 eventID, address validator) external view returns(bool);
      You, 2 weeks ago • Modifica nome da interface para ValidableEvent. I…
  /// @notice allows sender to reject an event. Should raise error if event was already validated.
  /// @param eventID Identifier of the event.
  /// @param rejectRate A rate that can be used by contracts to measure approval when needed.
  function rejectEvent(uint256 eventID, uint256 rejectRate) external;

  /// @notice allows sender to approve an event. Should raise error if event was already validated.
  /// @param eventID Identifier of the event.
  /// @param approvalRate A rate that can be used by contracts to measure approval when needed.
  function approveEvent(uint256 eventID, uint256 approvalRate) external;

  /// @notice should return the issuer of the event.
  /// @param eventID Identifier of the event.
  /// @return address of the issuer.
  function issuerOf(uint256 eventID) external view returns (address);

  // Event emmited when an event is created.
  event EventCreated(
    uint256 indexed eventID,
    address indexed issuer,
    address[] parties,
    address[] validators
  );

  // Event emmited when an event is approved.
  event EventApproved(
    uint256 indexed eventID,
    address indexed validator,
    uint256 approvalRate
  );

  // Event emmited when an event is revoked.
  event EventRejected(
    uint256 indexed eventID,
    address indexed validator,
    uint256 rejectionRate
  );
}
```

Figura 7. Código da interface ValidableEvent. Disponível no repositório em interfaces/IERCValidableEvent.sol

5. UTILIZAÇÃO DO CONTRATO INTELIGENTE

Um contrato na rede Ethereum possui um endereço específico e é possível interagir com ele através de chamadas de função. Funções que não modificam o estado da rede não tem nenhum custo de gas e podem ser respondidas prontamente pelo nó que recebe a chamada, já chamadas que alteram seu estado precisam ser assinadas com a chave privada do usuário e ser processada pela rede antes que o usuário possa obter uma resposta.

A interação com a rede Ethereum se dá por chamadas de método remotas (RPC) para um nó da rede. Alguns serviços fornecem APIs pelas quais é possível se comunicar com a rede, como a Infura [10], essas APIs agem como intermediários, apenas repassando as transações que são assinadas pelo usuário e retornando com a resposta da rede ou ID de transação para operações de escrita (Figura 8).

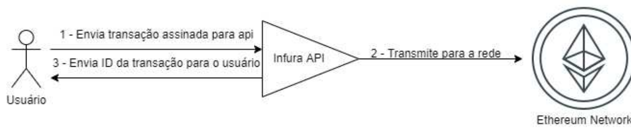

Figura 8. Interagindo com o contrato inteligente

Devido a complexidade da interação com um contrato para o usuário comum, sendo necessária a obtenção de uma chave de API e interação de baixo nível com a rede, um trabalho relacionado foi desenvolvido por Caio Sanches, com o intuito de prover uma interface web que facilita a interação com o contrato, ainda utilizando uma rede de testes pública, devido aos custos necessários para o deploy e utilização do contrato na rede oficial.

Para utilizar a interface web, é necessário utilizar um navegador web baseado em chromium com Metamask, uma extensão que permite que usuários realizem o gerenciamento de seus endereços na rede Ethereum de maneira segura, servindo de interface para autenticação e assinatura de transações [11].

As principais funcionalidades da interface web são verificar pagamentos onde o endereço é criador, recebedor ou verificador. Possibilita fácil interação com a rede para criar, validar e receber os valores referentes a pagamentos criados dentro do contrato (Figura 9). A interface possui uma página pública que permite que o usuário interaja com o contrato na rede de testes [12].

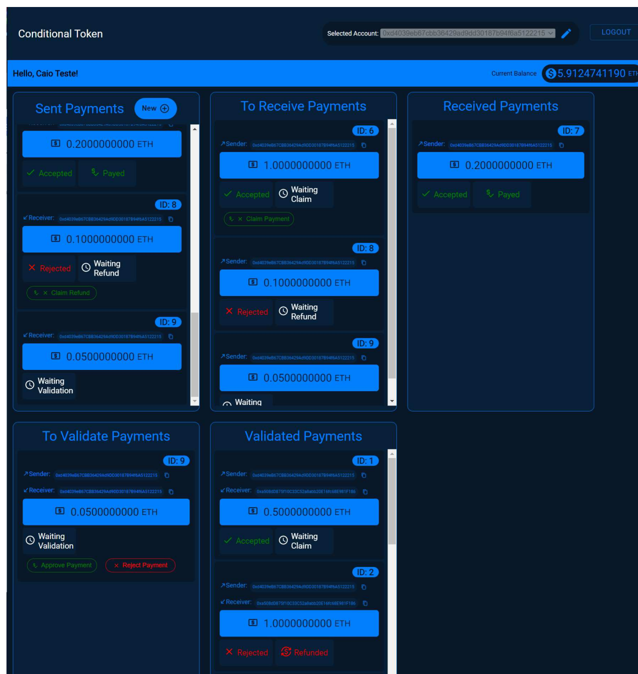

Figura 9. Visão Geral da Interface Web

O contrato com o qual a interface interage está na rede de testes Ropsten, uma rede Ethereum descentralizada onde é possível requisitar valores de Ethereum para interagir e testar contratos inteligentes antes de realizar a implantação do contrato na rede oficial. O contrato é identificado através do endereço 0xb0af2f3733E807de9C05A65f79E8532C8E1722e0 [13].

6. DESAFIOS

Se tratando de uma rede descentralizada onde cada nó possui uma cópia do estado da rede e é preciso entrar em consenso para modificá-lo, as interações com um contrato inteligente tem um custo atrelado e isso reflete na forma que sua modelagem de dados e até lógica é implementada. É preciso considerar que cada variável modificada representa custo adicional e pode inviabilizar o uso do contrato como um todo.

A interação com o contrato também se mostra um desafio, dado que para enviar requisições para a rede é necessário ter acesso a um nó, seja próprio ou de terceiros através de APIs públicas, bem como um endereço público na rede, que pode ser gerenciado por extensões como o Metamask. Mesmo utilizando uma interface web, a utilização de aplicações descentralizadas ainda se prova um desafio para o usuário comum.

7. CONCLUSÃO

As ferramentas disponíveis para o desenvolvimento de contratos inteligentes já estão bem avançadas, permitindo um desenvolvimento ágil e com frameworks bem estruturados para testes e implantação da aplicação.

O desenvolvimento deste modelo serviu para entender melhor as limitações da tecnologia e explorar o problema da inserção de dados não digitalizados como condição da execução de um contrato inteligente. A interface proposta visa tornar a validação do evento o mais genérica possível, permitindo que lógicas diferentes de validação sejam aplicadas para cada caso de uso.

O custo atrelado a utilização de contratos inteligentes se mostra um fator limitante, requerendo cautela com os algoritmos e estruturas de dados utilizados na execução de um contrato. É possível utilizar técnicas de empacotamento de variáveis, reduzindo variáveis que utilizam menos de 256 bytes dentro de uma única para reduzir o custo de transação.

A interface genérica proposta pode ser utilizada para desenvolvimento de outros casos de uso, de forma a validar e evoluir, podendo futuramente ser submetida como um padrão de desenvolvimento de contratos inteligentes da rede Ethereum.

8. TRABALHOS FUTUROS

O presente trabalho apresenta uma proposta de interface para virtualização de eventos do mundo real, nesse caso, a validação de um pagamento, mas ainda há muito espaço para testar essa interface em outros casos de uso, sendo passível de aprimoramento e evolução. A interface proposta neste trabalho pode servir de base para aplicações como uma seguradora descentralizada, onde validadores podem atestar o sinistro do carro e definir que porcentagem do valor é devido para o contratante do seguro, ou crowdfundings descentralizados, onde a verba pode ir sendo liberada para o organizador do projeto a medida que a comunidade atesta que os objetivos estão sendo cumpridos.

Sobre o contrato de pagamento seguro, é possível ainda aprimorar seu código com o intuito de reduzir custos de transação, como utilizar técnicas de empacotamento de variáveis e modelagem mais simples, guardando apenas os dados minimamente necessários para sua execução.

Por fim, uma forma de estender a usabilidade do contrato seria o desenvolvimento de um contrato inteligente que modele uma plataforma de reputação, auxiliando na escolha de validadores para os pagamentos seguros.

9. AGRADECIMENTOS

Agradeço inicialmente aos meus pais, Marcos Barros de Medeiros e Diana Karla Targino dos Santos Lima Barros de Medeiros, que me proporcionaram o dom da vida e todo o apoio necessário para o meu desenvolvimento na computação. Agradeço também a minhas irmãs, Camila Yasmine e Tamara Marjorie, por serem pontos de apoio e sempre estarem presentes, trazendo alegria para minha vida. A Mayara de Freitas Fernandes, minha namorada, por também acreditar, incentivar e preencher minha vida com amor e alegria. Agradeço a Tiago Lucas e ao Professor Antão por toda orientação e sugestões. Agradeço a meu amigo Caio Sanches, que topou realizar um trabalho relacionado, desenvolvendo uma interface web para a aplicação e me deu bastante apoio. Agradeço a Universidade Federal de Campina Grande, todo o corpo docente do curso de Ciências da Computação e todo o quadro de funcionários por toda a dedicação e esforço em proporcionar para a população um ambiente e educação de qualidade distinta. Por fim, agradeço aos amigos que fiz durante a graduação, por me fornecerem apoio e incentivo durante toda a jornada acadêmica, compartilhando das mesmas dores e celebrando cada vitória.

10. REFERENCIAS

[1] NAKAMOTO, Satoshi: Bitcoin whitepaper. 2008. Disponível em: https://bitcoin.org/bitcoin.pdf. Acesso em: 02 ago. 2022.

[2] CHAVEZ-DREYFUSS, Gertrude. Crypto market cap surges to record $2 trillion, bitcoin at $1.1 trillion. Reuters. 2021. Disponível em: https://www.reuters.com/article/us-crypto-currency-marketcap-idUSKBN2BS1I7. Acesso em: 02 ago. 2022

[3] SZABO, Nick. Smart Contracts: Building Blocks for Digital Markets. 1996. Disponível em: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html. Acesso em: 02 ago. 2022.

[4] BUTTERIN, Vitalik. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. 2014. Disponível em: https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf. Acesso em: 02 ago. 2022

[5] Tiago L.P. C, Joaquim J.C.M , J. Antão B.M et al. Contratos Inteligentes e o Usuário Comum: Revisão Sistemática e Agenda de Pesquisa. 2022. Disponível em: https://drive.google.com/file/d/1fkRSEGIRiyW1LUr7LHceYlLJ8Qav5myK. Acesso em: 17 ago. 2022.

[6] LIRA, Caio Sanches Batista. Uma interface web para um contrato inteligente. Trabalho de conclusão de curso, Universidade Federal de Campina Grande, Campina Grande, 2022.

[7] ROSA, João Roberto Conceição. Marketplace no Brasil: desafios, vantagens e tendências deste modelo de negócio para empresas varejistas. 2019. 68 f. Dissertação (Programa de Pós-Graduação Stricto Sensu em Administração de Empresas), Faculdade FIA, São Paulo, 2019. Disponível em: https://fia.com.br/wp-content/uploads/2019/05/Jo%C3%A3o-Roberto-Concei%C3%A7%C3%A3o-Rosa_Vers%C3%A3o-Final_MPROF4.pdf . Acesso em: 02 ago. 2022.

[8] BENIICHE, Abdeljalil: A Study of Blockchain Oracles. 2020. Disponível em: https://arxiv.org/pdf/2004.07140.pdf. Acesso em: 16 ago. 2022.

[9] WOOD, Gavin: Ethereum: a secure decentralized generalized transaction ledger EIP-150 revision. Disponível em: http://gavwood.com/paper.pdf. Acesso em: 16 ago. 2022.

[10] Infura. Disponível em: https://infura.io/. Acesso em: 16 ago. 2022.

[11] Metamask. Disponível em: https://metamask.io/. Acesso em: 16 ago. 2022.

[12] Interface Web Pagamento Seguro. Disponível em: https://conditional-token-frontend.herokuapp.com/ . Acesso em: 16 ago. 2022.

[13] Endereço do pagamento seguro. Disponível em: https://ropsten.etherscan.io/address/0xb0af2f3733e807de9c05a65f79e8532c8e1722e0 . Acesso em: 16 ago. 2022.