Uma Abordagem Dirigida por Modelo para Desenvolvimento de Contratos Inteligentes na Ethereum Virtual Machine

Gislainy Crisostomo Velasco, Sergio T. Carvalho

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 131 — 74001-970 — Goiânia — GO -– Brasil

gislainycrisostomo@discente.ufg.br, sergio@inf.ufg.br

Abstract. *Blockchain technology enables the implementation of smart contracts that express in code form the clauses of real-world contracts so that they can be executed without the need of an intermediary. In this context, decentralized applications (DApps) have been introduced that utilize features such as immutability, decentralization, transparency, and privacy. However, the development of DApps is limited to the scope of contracts and the specifics of blockchain. Developers need to understand the entire environment inherent in the technology, such as contract programming language issues, security issues, and others. The goal of this paper is to define a metamodel that makes it easier for both domain experts and developers to model contracts at a high level. The proposal consists of a metamodel that abstracts the technical issues inherent to the Ethereum Virtual Machine (EVM) that allows defining the essential elements that must be implemented in the final contract.*

Resumo. *A tecnologia blockchain possibilita a implementação de contratos inteligentes que expressam na forma de código as cláusulas de contratos do mundo real de modo a serem executadas sem a necessidade de um intermediário. Nesse contexto, foram introduzidas as aplicações decentralizadas (DApps) que utilizam recursos, como imutabilidade, descentralização, transparência e privacidade. Contudo, o desenvolvimento de DApps é limitado ao escopo de contratos e às especificidades da blockchain. Os desenvolvedores necessitam compreender todo o ambiente inerente à tecnologia, como questões da linguagem de programação dos contratos, questões de segurança e outros. O objetivo deste artigo é definir um metamodelo que facilite tanto para os especialistas do domínio quanto para desenvolvedores a modelagem dos contratos em alto nível. A proposta consiste em um metamodelo que abstraia as questões técnicas inerentes à Ethereum Virtual Machine (EVM) que permita definir os elementos essenciais que devem ser implementados no contrato final.*

1. Introdução

A tecnologia *blockchain* teve uma ampla adoção devido às suas características de imutabilidade, descentralização, transparência e privacidade [Angelis and Ribeiro da Silva 2019, Khan et al. 2019]. Essas características têm permitido que diversas áreas (*e.g.,* setor financeiro, cadeia de suprimentos, registros médicos de saúde, internet das coisas) agreguem novos valores [Angelis and Ribeiro da Silva 2019]. A tecnologia é divida em quatro estágios, sendo que cada estágio possui uma proposta específica que possibilita a criação de novos mercados e formas de interação na criação de valor.

O primeiro estágio, conhecido como a primeira geração de blockchain (Blockchain 1.0) iniciou-se com a proposta de Satoshi Nakamoto [Nakamoto 2008] que permitia confiança entre partes desconhecidas sem a necessidade de outra entidade intermediária. O segundo estágio (Blockchain 2.0) foi iniciado com a introdução de contratos inteligentes, ou simplesmente contratos [Szabo 1996]. Os contratos inteligentes permitem que os contratos do mundo real sejam expressos por código de computador e que a sua execução seja descentralizada em uma blockchain. Dessa forma, quando as condições do contrato são verdadeiras, qualquer parte poderia reivindicar sua parte do acordo, sem a hesitação ou necessidade de um intermediário [Buterin et al. 2014, Khan et al. 2019, Rajasekaran et al. 2022, Wang et al. 2019].

As aplicações descentralizadas (do inglês, *Decentralized Applications* – DApps) são consideradas como o terceiro estágio (Blockchain 3.0). DApps são aplicações que executam de forma decentralizada em uma rede de computadores ponto-a-ponto (do inglês, *Peer-to-Peer* – P2P). Por fim, o mais recente estágio e emergente (Blockchain 4.0) envolve a inclusão de inteligência artificial (IA) na tecnologia blockchain para auxiliar nas tomadas de decisão [Angelis and Ribeiro da Silva 2019].

Dado esse cenário, existem desafios para o desenvolvimento de contratos e DApps. Para os contratos, há necessidade de entendimento da estrutura e linguagem da blockchain. Em blockchain baseada na Ethereum Virtual Machine (EVM), os desenvolvedores necessitam compreender a linguagem de programação Solidity ou Viper, enquanto que para a plataforma Hyperledger Fabric é possível utilizar-se linguagens de programação de propósito geral, como Java [Chirtoaca et al. 2020], e para a Solana, a linguagem Rust. Além disso, características subjacentes da blockchain impactam diretamente no desenvolvimento das DApps. Após a implantação de um contrato, não há a possibilidade de modificação, devido à imutabilidade, propriedade inerente à tecnologia. Nesse sentido, para a EVM existem soluções, como o uso de *Proxy* (interceptador de chamadas) que possibilita a alteração da implementação para evoluções ou correções de erros. Porém, há uma série de questões para se considerar na implementação de proxy [Nadolinski and Spagnuolo 2018].

Questões de segurança e vulnerabilidade em contratos e DApps são as mais exploradas na literatura [Atzei et al. 2017, Chen et al. 2020]. As falhas de segurança presentes nos contratos permitem que invasores drenem fundos de contratos causando perda de milhões de dólares. Em 2016, por meio de um ataque em um contrato DAO[1], um invasor explorou a vulnerabilidade de reentrância e roubou cerca de US$ 60 milhões, levando ao *hard fork* da Ethereum, ou seja, à rede se dividiu em duas blockchains.

Diante desse cenário, os desenvolvedores necessitam compreender todo o ambiente inerente à tecnologia, como especificidades da blockchain, questões da linguagem de programação dos contratos, questões de segurança e outros. Para vários autores, a concepção de um modelo de alto nível seguindo os padrões da engenharia de software facilitaria o desenvolvimento de contratos para blockchain [Ben Slama Souei et al. 2021, Chirtoaca et al. 2020, Guida and Daniel 2019, Hamdaqa et al. 2020]. Nesse caminho, existem algumas propostas com foco na Engenharia Dirigida por Modelo (do inglês,

[1]Uma organização autônoma descentralizada (do inglês, *Decentralized Autonomous Organization* – DAO) são organizações que fazem o uso de contratos inteligentes na sua governança e permitem que os seus usuários tomem decisões de forma descentralizada.

Model-driven Engineering – MDE) para construção de contratos, desde sua concepção até a geração de código para a plataforma correspondente [Hamdaqa et al. 2020, Santiago et al. 2021, Ben Slama Souei et al. 2021]. O objetivo deste artigo é definir um metamodelo que facilite tanto para os especialistas do domínio quanto para desenvolvedores a modelagem dos contratos em alto nível. A proposta consiste em um metamodelo que abstraia as questões técnicas inerentes à Ethereum Virtual Machine (EVM) que permita definir os elementos essenciais que devem ser implementados no contrato final.

O artigo está organizado em outras 4 (quatro) seções, além desta introdutória. A Seção 2 apresenta os conceitos que norteiam esta pesquisa, como Blockchain, Contratos Inteligentes, DApps e MDE. Na Seção 3 são apresentados os trabalhos que utilizam MDE para facilitar o desenvolvimento de contratos. Em seguida, na Seção 4 é apresentada a proposta inicial do metamodelo para desenvolvimento de contratos. Por fim, na Seção 5 são apresentadas as considerações finais deste artigo e os próximos passos na pesquisa.

2. Referencial Teórico

2.1. Blockchain

O conceito de Blockchain foi utilizado por Satoshi Nakamoto [Nakamoto 2008] em seu *whitepaper* que apresenta a criptomoeda Bitcoin que teve uma ampla adoção, inicialmente em setores financeiros por resolver o problema de gasto duplo [Vujičić et al. 2018]. Posteriormente, a tecnologia foi empregada em diversos campos (*e.g* cadeia de suprimento, registro de saúde, serviços públicos) devido às suas características de descentralização, persistência, anonimato, transparência, imutabilidade, integridade e auditabilidade [Zheng et al. 2017, Tijan et al. 2019].

Um blockchain consiste em uma lista encadeada de blocos, onde todos os blocos são vinculados ao bloco anterior por meio do seu *hash* formando uma cadeia protegida por criptografia [Zheng et al. 2017, Seebacher and Schüritz 2017, Francisco and Swanson 2018, Tijan et al. 2019]. Os blocos são compostos por transações e por um contador de transações. A quantidade de transações em um bloco depende do tamanho dele. As transações são autenticadas usando mecanismo de criptografia assimétrica [Hellman et al. 1976].

Para que um bloco seja anexado à cadeia de bloco, cada blockchain adota um algoritmo de consenso para que os nós da rede cheguem a um consenso do estado da blockchain. Existem vários algoritmos de consenso com técnicas diferentes, *e.g.,* prova matemática, processo de eleição. Os algoritmos de consenso mais utilizados são prova de trabalho (do inglês, *Proof of Work* – PoW)[2] e prova de participação (do inglês, *Proof of Stake* – PoS)[3].

As blockchains são classificadas em três tipos: públicas, de consórcios e privadas [Zheng et al. 2017]. Nas blockchain públicas as informações são visíveis ao público e qualquer indivíduo pode participar dela, seja enviando transações ou participando do processo de consenso. Nas de consórcios, somente os nós pré-selecionados podem participar

[2]Requer que seja encontrado um valor (*nonce*) que seja menor igual ao determinado valor. Esse algoritmo utiliza alto poder computacional.

[3]Os nós participantes precisam ter uma certa quantidade de moedas para participar da rede. Para gerar um novo bloco o minerador deve enviar para si mesmo uma quantidade de moedas provando sua posse.

do processo de consenso. Contudo, nas privadas somente os nós de uma organização podem participar. Nas de consórcio e privadas, as permissões de gravação podem ser públicas ou restritivas.

As blockchains públicas são consideradas totalmente descentralizadas e utilizam incentivos econômicos[4], *e.g.,* Ethereum, Polygon. As de consórcios são consideradas parcialmente descentralizadas e as privadas são consideradas centralizadas. As privadas são empregadas em contextos em que há necessidade que os dados sejam auditáveis e que somente entidades específicas possam usar as informações, *e.g.,* na saúde.

2.2. Contratos Inteligentes

O conceito de contratos inteligentes foi introduzido por Nick Szabo [Szabo 1996] em 1996 e somente foi possível sua implementação com introdução da tecnologia blockchain [Wang et al. 2019]. Um contrato inteligente pode ser definido como um protocolo de computador onde as partes entram em comum acordo e quando as condições desse acordo são verdadeiras as cláusulas do contrato inteligentes definidas em código são executadas sem a necessidade de uma terceira parte confiável [Buterin et al. 2014, Rajasekaran et al. 2022, Wang et al. 2019].

Como os contratos são normalmente executados em uma blockchain, após a implementação do código-fonte não é possível alteração, pois um contrato inteligente é imutável. Assim, as partes precisam entrar em acordo antes da sua implantação. Após sua implantação, não há necessidade de confiança entre as partes e nem a presença de um terceiro confiável e todas alterações podem ser rastreadas pelas partes interessadas. Um contrato cria um conjunto de funcionalidades confiáveis para todos os usuários envolvidos [Cai et al. 2018].

A Ethereum foi a primeira blockchain a aceitar contratos inteligentes complexos, seguida por outras plataformas, *e.g.,* Hyperledger, Solana. O Bitcoin aceita contratos inteligentes básicos, porém sem lógica complexa devido a limitações da linguagem de *script* usada na plataforma [Wang et al. 2019]. A Ethereum é a plataforma para desenvolvimento de contratos mais popular devido à EVM que é uma máquina de estado distribuído Turing-completo [Gramlich 2020, Wang et al. 2019]. A EVM suporta as linguagens como Solidity e Vyper. A Solidity é a linguagem de programação mais popular no desenvolvimento de contratos [Gramlich 2020, Kaleem et al. 2020].

2.2.1. Ciclo de vida do contrato

Para [Wang et al. 2019] o ciclo de vida de um contrato é resumido em cinco etapas: 1) negociação; 2) desenvolvimento; 3) implantação; 4) manutenção; e 5) aprendizagem e autodestruição. Nesse trabalho o foco está na etapa de 2.

Na etapa de desenvolvimento há uma série de ferramentas disponíveis que auxiliam o desenvolvedor na codificação e testes (*e.g.,* Hardhat, Truffle Suite, Remix Ethereum IDE). O ambiente de execução impacta diretamente na modelagem dos contratos, pois diferentes algoritmos de consenso impactam no custo das operações [Wang et al. 2019].

[4]Uma forma de incentivar que os minerados validem as transações seguindo o mecanismo de consenso sem manipulação da rede.

Essa etapa é fundamental na concepção de um contrato sendo necessária uma análise de segurança para possíveis vulnerabilidades no contrato – em muitos casos há o armazenamento de milhões de dólares. Além disso, há o fato de que possíveis erros não detectados nessa etapa têm um caráter irreversível, devido à natureza imutável dos contratos. Na EVM, as propostas de implementação EIP-897 e EIP-1967 adicionam capacidade de atualização do contrato através do mecanismo de *proxy* que permite ter um contrato com estado consistente e um contrato lógico ou de implementação, sendo que o código-fonte reside no contrato de implementação e toda chamada é encaminhada para ele.

Após a etapa de modelagem e desenvolvimento, os contratos são implantados por uma transação de criação de contrato, essa transação é composta pelo *bytecode*[5]. Após a transação ser validada pelos minerados, o *bytecode* é armazenado em um bloco específico e um endereço é associado ao contrato. Finalizada esta etapa, um contrato está pronto para uso.

Um contrato implantado possui um estado e para esse estado ser alterado é necessária a submissão de uma nova transação. Existem dois tipos de métodos na blockchain que lidam com estados: (1) de leitura; (2) de modificação. Os métodos de leitura são usados para recuperar o estado do contrato e podem ser invocados sem nenhum custo. Os métodos que modificam o estado da cadeia de bloco são assinados criptograficamente e têm um custo envolvido, pois será necessário que um minerador valide a transação. Geralmente quando há modificações no estado do contrato são usados os eventos que agem como um histórico e auxiliam as aplicações fora da cadeia no rastreio das informações armazenadas ao longo do ciclo de vida do contrato. As aplicações fora da cadeira necessitam possuir a interface binária de aplicação (do inglês, *Application Binary Interface* – ABI) que contém os protótipos das funções e eventos utilizadas na interação dos contratos.

2.3. Aplicações Descentralizadas

Devido a sua natureza descentralizada, o ecossistema da blockchain possibilitou o desenvolvimento de DApps, aplicações que executam em uma rede de computadores P2P, como, por exemplo, BitTorrent e BitMessage [Wu et al. 2021]. No contexto da blockchain, um DApp deveria armazenar e manipular todas as suas informações na blockchain, porém devido às limitações atuais (*e.g.,* taxa de transferência, latência, custo) somente uma parte das funcionalidades que necessitam ser resistentes a adulterações ou rastreáveis são implementadas *on-chain* [Cai et al. 2018, Wu et al. 2021].

As arquiteturas de DApps estão limitadas a contratos. [Wu et al. 2021] propuseram 3 (três) categorias de arquiteturas: (A) os DApp clientes que interagem diretamente com os contratos, por exemplo, utilizando um Metamask[6] (Figura 1 A); (B) os que tem um servidor centralizado que interagem com blockchain (Figura 1 B) e; (C) os híbridos (Figura 1 C).

Idealmente, os DApps deveriam ser totalmente hospedados na blockchain para que os usuários finais conseguissem utilizar sem a necessidade de manutenção dos de-

[5]É a linguagem de baixo nível interpretada pela EVM. Os *bytecodes* são compilados a partir das linguagens de alto nível, como Solidity ou Vyper.

[6]Software que gerencia as contas de usuários e permite a interação com blockchain EVM. Para mais informações acesse https://metamask.io/

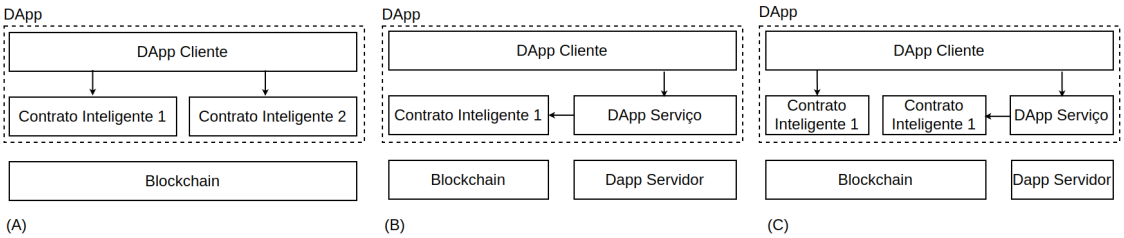

Figura 1. Três arquiteturas de DApps: (A) comunicação direta, (B) comunicação indireta e (C) hibrída. Adaptado de [Wu et al. 2021]

senvolvedores iniciais. Com isso, algumas caraterísticas são comumente aceitas para um Dapps: seu código-fonte deveria ser aberto para que a comunidade possa manter e evoluir; os dados devem ser armazenados em uma blockchain; ter suporte a token e suas recompensas devem ser por meio de token; ter um consenso descentralizado e; nenhum ponto de falha [Cai et al. 2018, Swan 2015].

Para [Cai et al. 2018] a construção de DApps futuros deve ter um melhor desempenho (*e.g.,* diminuir a latência, aumentar a taxa de transferência para suportar volume grande de usuários), taxas de transações mais baixas, manutenção mais flexível. Entretanto, todos esses detalhes estão atrelados à blockchain escolhida, com isso, é essencial que os desenvolvedores considerem no momento da modelagem qual tecnologia adotar.

2.4. Engenharia Dirigida por Modelo (Model-driven Engineering)

Model-Driven Engineering (MDE) considera os modelos como artefatos primários no desenvolvimento de software [Seidewitz 2003]. Os modelos permitem que profissionais técnicos e não técnicos compartilhem conhecimento e tenham melhor facilidade na comunicação. Um modelo é uma abstração de um sistema em estudo [Rodrigues da Silva 2015, Seidewitz 2003]. Os metamodelos são a especificação de um modelo, ou seja, um metamodelo conduz como um modelo deve ser expresso.

Um problema na metamodelagem é a definição de um modelo inicial, pois para definir um metamodelo é necessário que exista um meta-metamodelo descrevendo a linguagem de modelagem [Rodrigues da Silva 2015]. Com isso a Object Management Group (OMG) propõe a MetaObject Facility (MOF) baseada em uma arquitetura de quatro camadas em que cada elemento inferior é instância de um elemento superior [Seidewitz 2003, Vieira et al. 2016]. As camadas da MOF são:

- M3: representação do meta-metamodelo. É a responsável por definir a linguagem para especificar o metamodelo. A sua instância é definida por si, ou seja, MOF é definido em MOF sem necessidade de um nível superior. Os metamodelos desse trabalho são instâncias do meta-metamodelo Ecore – baseado no MOF.
- M2: representam os metamodelos. Nessa camada os metamodelos são a instanciação dos meta-metamodelos.
- M1: representam os modelos. Os modelos são definidos conforme os metamodelos presente no M2.
- M0: contêm as instâncias ou objetos definidos no modelo.

2.4.1. Eclipse Modeling Framework

Eclipse Modeling Framework (EMF) é um *framework* para modelagem integrado ao Eclipse [Vieira et al. 2016]. O EMF fornece uma estrutura em tempo de execução que permite a modelagem e validação utilizando uma interface de usuário e geração de códigos a partir de modelos [Steinberg et al. 2008]. O meta-metamodelo usado para representar modelos na EMF é a Ecore (Figura 2). A seguir é apresentado um detalhamento dos elementos básicos do Ecore.

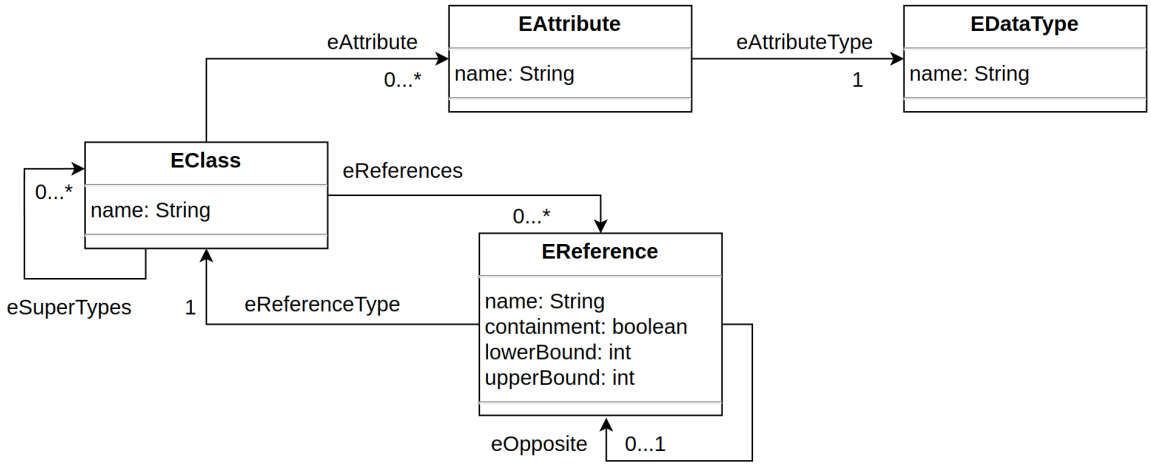

Figura 2. Componentes básicos do meta-metamodelo Ecore. Adaptado de [Steinberg et al. 2008].

- *EClass*: são responsáveis por modelar as classes; sua identificação é através do nome. As classes suportam um ou mais atributos (*EAttribute*) e podem referenciar outras classes (*EReference*). A herança também é permitida por meio das referências com supertipos (*eSuperTypes*).
- *EAttribute*: modela os atributos de um objeto ou componentes de dados. São identificados por um nome com um tipo associado.
- *EDataType*: usado para modelar os tipos cuja estrutura não é modelada como classe. São identificados por um nome e são usados como tipos de atributos.
- *EReference*: são usados nas associações entre as classes. São identificados pelo nome e sua associação a uma classe. Os limites inferiores e superiores são especificados para indicar sua multiplicidade. No EMF as associações de composição são denotadas usando o tipo *containment*.

O EMF foi utilizado neste trabalho como abordagem para modelagem do metamodelo proposto por ser um projeto de código aberto que oferece uma ferramenta integrada que permite a modelagem, validação e geração de código [Steinberg et al. 2008]. Foi usada a ferramenta Obeo Designer que utiliza as bases da tecnologia EMF com integração com Acceleo e a Sirius[7] para elaboração completa de metamodelos seguindo a Ecore.

[7]https://wiki.eclipse.org/Sirius

3. Trabalhos Relacionados

A abordagem utilizando técnicas MDE está sendo empregada por vários autores para modelar um contrato em alto nível [Ben Slama Souei et al. 2021, Chirtoaca et al. 2020, Guida and Daniel 2019, Hamdaqa et al. 2020, Santiago et al. 2021].

As principais abordagens identificadas são: notação de modelagem de processos de negócio (do inglês, *Business Process Model and Notation* – BPMN) [Corradini et al. 2022]; linguagem de modelagem unificada (do inglês, *Unified Modeling Language* – UML); programação por blocos [Guida and Daniel 2019], [Achour et al. 2021, Garamvölgyi et al. 2018, Jurgelaitis et al. 2022]; e metamodelo [Hamdaqa et al. 2020, Ben Slama Souei et al. 2021, Jurgelaitis et al. 2022].

[Ben Slama Souei et al. 2021] propõem uma extensão do modelo unificado de descrição de serviços por meio de um metamodelo que define uma linguagem de descrição uniforme para contratos. O objetivo da proposta é fornecer aos usuários finais um mecanismo para compreender os contratos implantados. O metamodelo proposto permitem ter uma visão operacional, técnica e de negócios, porém não auxilia na concepção e desenvolvimento do contrato. A visão operacional fornecida somente permitem uma visão das assinaturas das funções e eventos do contrato implantado.

[Hamdaqa et al. 2020] propõem a IContractML para modelagem de contratos com suporte para as plataformas IBM Hyperledger Composer, Azure Blockchain Workbench e Ethereum. Entretanto, o código gerado não é o suficiente para implementação de um contrato funcional por possuir somente a estrutura e não o comportamento. Diferentemente, este trabalho tem como objetivo definir um metamodelo com foco na EVM que possibilite a implementação do contrato.

4. Metamodelo

O metamodelo proposto (Figura 3) incorpora os elementos essenciais para o desenvolvimento de um contrato. Sua modelagem foi inspirada na estrutura básica de um contrato na EVM.

Contract é o elemento principal do metamodelo, a partir do qual é possível especificar os elementos que o compõem. Os elementos podem ser do escopo do contrato (*e.g,* variáveis, funções, construtores, *structs*) ou outros contratos. A capacidade de reuso dos contratos por meio da herança permite que os desenvolvedores aproveitem contratos amplamente testados pela comunidade (*e.g.* OpenZeppelin) ou divida as responsabilidades em contratos menores.O metamodelo proposto incorporou essa possibilidade.

Um contrato é uma abstração do mundo real em código imutável e rastreável, com isso é fundamental que haja uma representação das propriedades do contexto, seja por variáveis de estado, seja por funções. As variáveis de estado são representadas no escopo do contrato e podem ser modificadas no momento que um contrato é instanciado ou posteriormente por uma função. As funções são a representação das regras no momento da modelagem do contrato e são as responsáveis por atender os acordos estabelecidos entre as partes. Elas podem assumir comportamento simples, por exemplo, somente atendendo as condições definidas nas variáveis de estado (determinística) ou sua execução depender das entradas recebidas. Quando uma regra do contrato é cumprida, é possível notificar usando os eventos que uma determinada ação ocorreu e DApps interes-

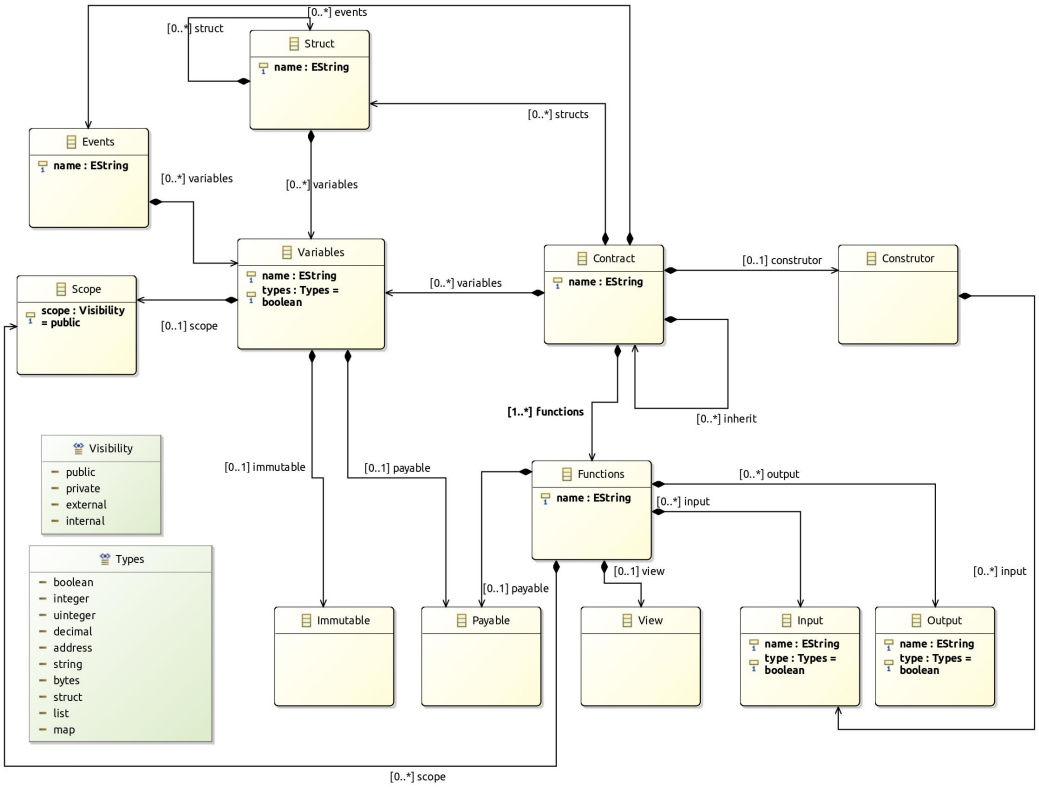

Figura 3. Metamodelo proposto.

sados podem executar suas regras. Esses eventos podem ser estruturados seguindo uma nomenclatura diferente das variáveis de estado e podem ser utilizados para efeito de auditoria. Os eventos são muito utilizados pelas aplicações servidoras de DApps (veja Seção 2.3).

Cada elemento do metamodelo proposto é descrito a seguir.

- ***Contract***: elemento raiz do metamodelo.
- ***Variables***: representam o estado do contrato e seus valores são armazenados permanentemente. São representadas no nível do contrato e podem ser modificadas quando o contrato é instanciado ou na invocação remota de uma função (*Remote Procedure Call*, RPC). Variáveis imutáveis (representadas pela composição *Immutable*) não podem ser modificadas após sua inicialização. Em caso contrario, todas as variáveis podem ser modificadas posteriormente em chamadas RPC. As variáveis são compostas por um nome e tipo. O tipo depende da linguagem utilizada para escrever o contrato. Por exemplo, na EVM existe o tipo *address* utilizado para representar os endereços de contas ou contratos e pode lidar com a transferência de moeda (*Payable*). As variáveis possuem, ainda, escopo de visibilidade (*Scope*); caso não seja definido, é atributo o escopo padrão da linguagem utilizada.
- ***Struct***: representação de uma estrutura mais complexa sendo declarada no nível do contrato. Pode ser utilizada como um tipo quando declarada uma variável de estado ou de escopo de uma função. Uma *struct* deve possuir pelo menos um membro, podendo ser composta por outras *structs*.

- ***Constructor***: similar a uma classe na modelagem orientada a objetos, as variáveis de estado podem ser inicializadas quando o contrato é instanciado. Na EVM somente pode existir um *constructor* por contrato.
- ***Functions***: essenciais na construção de um contrato, são responsáveis por verificar as condições e executar quando são verdadeiras. Caso contrário, sua execução é revertida e o estado do contrato não é alterado. As funções podem ser dividas em dois grupos: (1) de visualização; (2) de mutação. As funções de visualização (*View*) não alteram o estado do contrato e sua invocação não possui custo. As funções de mutação podem alterar o estado do contrato caso sua lógica de execução seja verdadeira e possui um custo envolvido que dependerá da blockchain. Além disso, as funções de mutação podem receber transferência de moeda (*Payable*). As funções podem possuir entradas (*Input*) e saídas (*Output*). Por padrão, as de visualização sempre possuem uma ou mais saídas e as funções de mutação geralmente não têm saídas, pois sua execução é assíncrona; será somente retornada quando for chamada por outro contrato. Por fim, o escopo de visibilidade da função pode ser definido ou será utilizado o padrão da linguagem.
- ***Events***: responsáveis por registrar informações que podem ser consumidas para efeito de rastreabilidade. São responsáveis por facilitar o consumo de informações *on-chain* de forma estruturada.

As estruturas como listas ou mapas dependem da linguagem utilizada para construir o contrato. Dessa forma, quando for necessário representar alguma variável, seja de escopo do contrato, seja de entrada e saída de função, será necessária a modificação do esqueleto do contrato gerado.

5. Considerações Finais

A tecnologia blockchain permite a construção de aplicações descentralizadas por meio dos contratos inteligentes. As DApps utilizam os recursos da blockchain como, imutabilidade, descentralização, transparência e privacidade. Contudo, o desenvolvimento de DApps é limitado ao escopo de contratos e às especificidades da blockchain. Os desenvolvedores necessitam compreender todo o ambiente inerente à tecnologia, como questões da linguagem de programação dos contratos, questões de segurança e outros.

Diante desse cenário, esse artigo teve como propósito desenvolver um metamodelo para facilitar o desenvolvimento de contratos para EVM. O metamodelo proposto permite definir alto nível os elementos essenciais que devem ser implementados no contrato final. Após essa etapa, o próximo passo consiste em realizar a transformação do modelo em código para a linguagem Solidity.

Referências

Achour, I., Idoudi, H., and Ayed, S. (2021). Automatic generation of access control for permissionless blockchains: Ethereum use case. In *2021 IEEE 30th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 45–50.

Angelis, J. and Ribeiro da Silva, E. (2019). Blockchain adoption: A value driver perspective. *Business Horizons*, 62(3):307–314.

Atzei, N., Bartoletti, M., and Cimoli, T. (2017). A survey of attacks on ethereum smart contracts sok. In *Proceedings of the 6th International Conference on Principles of Security and Trust - Volume 10204*, page 164–186, Berlin, Heidelberg. Springer-Verlag.

Ben Slama Souei, W., El Hog, C., Sliman, L., Ben Djemaa, R., and Ben Amor, I. A. (2021). Towards a uniform description language for smart contract. In *2021 IEEE 30th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 57–62.

Buterin, V. et al. (2014). A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1.

Cai, W., Wang, Z., Ernst, J. B., Hong, Z., Feng, C., and Leung, V. C. M. (2018). Decentralized applications: The blockchain-empowered software system. *IEEE Access*, 6:53019–53033.

Chen, H., Pendleton, M., Njilla, L., and Xu, S. (2020). A survey on ethereum systems security: Vulnerabilities, attacks, and defenses. *ACM Comput. Surv.*, 53(3).

Chirtoaca, D., Ellul, J., and Azzopardi, G. (2020). A framework for creating deployable smart contracts for non-fungible tokens on the ethereum blockchain. In *2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 100–105.

Corradini, F., Marcelletti, A., Morichetta, A., Polini, A., Re, B., and Tiezzi, F. (2022). Engineering trustable and auditable choreography-based systems using blockchain. *ACM Trans. Manage. Inf. Syst.*, 13(3).

Francisco, K. and Swanson, D. (2018). The supply chain has no clothes: Technology adoption of blockchain for supply chain transparency. *Logistics*, 2(1):2.

Garamvölgyi, P., Kocsis, I., Gehl, B., and Klenik, A. (2018). Towards model-driven engineering of smart contracts for cyber-physical systems. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 134–139.

Gramlich, B. (2020). Smart contract languages: A thorough comparison. *ResearchGate Preprint*.

Guida, L. and Daniel, F. (2019). Supporting reuse of smart contracts through service orientation and assisted development. In *2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*, pages 59–68.

Hamdaqa, M., Metz, L. A. P., and Qasse, I. (2020). Icontractml: A domain-specific language for modeling and deploying smart contracts onto multiple blockchain platforms. In *Proceedings of the 12th System Analysis and Modelling Conference*, SAM '20, page 34–43, New York, NY, USA. Association for Computing Machinery.

Hellman, M., Merkle, R., Schroeppel, R., Washington, L., Diffie, W., and Schweitzer, P. (1976). *Results of an initial attempt to cryptanalyze the NBS Data Encryption Standard*.

Jurgelaitis, M., čeponienė, L., and Butkienė, R. (2022). Solidity code generation from uml state machines in model-driven smart contract development. *IEEE Access*, 10:33465–33481.

Kaleem, M., Mavridou, A., and Laszka, A. (2020). Vyper: A security comparison with solidity based on common vulnerabilities. In *2020 2nd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS)*, pages 107–111.

Khan, A. G., Zahid, A. H., Hussain, M., Farooq, M., Riaz, U., and Alam, T. M. (2019). A journey of web and blockchain towards the industry 4.0: An overview. In *2019 International Conference on Innovative Computing (ICIC)*, pages 1–7.

Nadolinski, E. and Spagnuolo, F. (2018). Proxy Patterns. https://blog.openzeppelin.com/proxy-patterns. Acessado em 02 de Junho de 2022.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260.

Rajasekaran, A. S., Azees, M., and Al-Turjman, F. (2022). A comprehensive survey on blockchain technology. *Sustainable Energy Technologies and Assessments*, 52:102039.

Rodrigues da Silva, A. (2015). Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems Structures*, 43:139–155.

Santiago, L., Abijaude, J., and Greve, F. (2021). A framework to generate smart contracts on the fly. In *Proceedings of the XXXV Brazilian Symposium on Software Engineering*, SBES '21, page 410–415, New York, NY, USA. Association for Computing Machinery.

Seebacher, S. and Schüritz, R. (2017). Blockchain technology as an enabler of service systems: A structured literature review. In *International conference on exploring services science*, pages 12–23. Springer.

Seidewitz, E. (2003). What models mean. *IEEE Software*, 20(5):26–32.

Steinberg, D., Budinsky, F., Merks, E., and Paternostro, M. (2008). *EMF: eclipse modeling framework*. Pearson Education.

Swan, M. (2015). *Blockchain: Blueprint for a new economy*. "O'Reilly Media, Inc.".

Szabo, N. (1996). Smart contracts: building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought,(16)*, 18(2):28.

Tijan, E., Aksentijević, S., Ivanić, K., and Jardas, M. (2019). Blockchain technology implementation in logistics. *Sustainability*, 11(4).

Vieira, M. A. et al. (2016). Modelagem de espaços inteligentes pessoais e espaços inteligentes fixos no contexto de cenários de computação ubíqua.

Vujičić, D., Jagodić, D., and Ranđić, S. (2018). Blockchain technology, bitcoin, and ethereum: A brief overview. In *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 1–6.

Wang, S., Ouyang, L., Yuan, Y., Ni, X., Han, X., and Wang, F.-Y. (2019). Blockchain-enabled smart contracts: Architecture, applications, and future trends. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(11):2266–2277.

Wu, K., Ma, Y., Huang, G., and Liu, X. (2021). A first look at blockchain-based decentralized applications. *Software: Practice and Experience*, 51(10):2033–2050.

Zheng, Z., Xie, S., Dai, H., Chen, X., and Wang, H. (2017). An overview of blockchain technology: Architecture, consensus, and future trends. In *2017 IEEE International Congress on Big Data (BigData Congress)*, pages 557–564.