Blockchain e Desenvolvimento Orientado a Modelos: Gerando o código de Smart Contract Automaticamente

TDC Recife

Claudia Rejane Oliveira Gomes
crog@cesar.org.br

10.out.2019

Quem sou eu

Claudia Rejane Oliveira Gomes
Engenheira de Software

Aluna do Mestrado Profissional em
Engenharia de Software da Cesar School

Orientador: Felipe Ferraz

Co-Orientador: Leandro Marques


C.E.S.A.R
centro de estudos e sistemas
avançados do recife

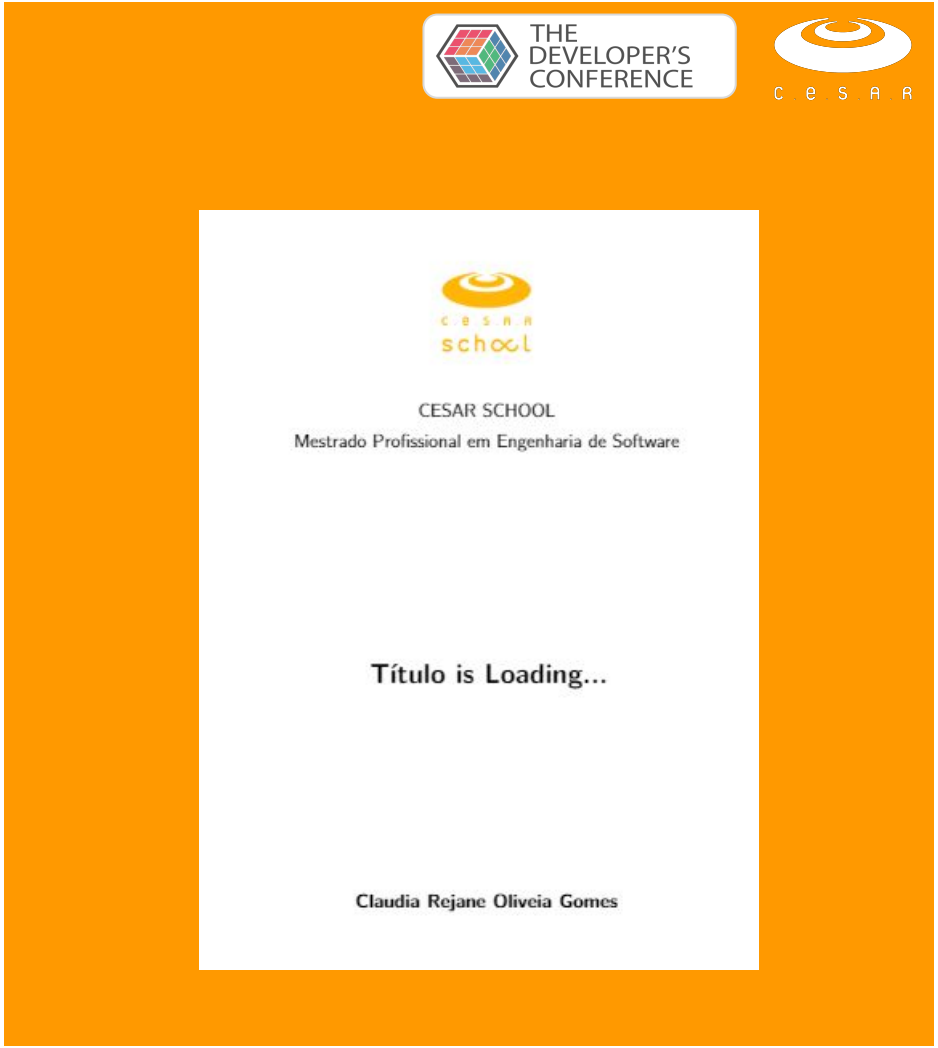
C.E.S.A.R

CESAR

Centro de Inovação que utiliza Design e Tecnologia da Comunicação e Informação para resolver problemas complexos de um mercado muito diverso.

Em torno de 650 colaboradores espalhados em alguns estados do país.

Primeiramente, um pouco de contexto



Nossa missão:

Construção de uma linguagem visual para a escrita de contratos

- Fácil de entender e utilizar
- Geração de código Semi-Automático/Automático
- Legível para pessoas de fora da área de Ti utilizar
- Coerente com o escopo do problema

O que é um contrato?

Um acordo jurídico entre duas ou mais partes, escrito e assinado por todos os envolvidos no acordo;

É composto por todas as condições que foram acordadas entre as partes e que devem ser cumpridas.

Problemas

- A validação depende de terceiros e está sujeita a um sistema judicial público
- Linguagem jurídica é passível de múltiplas interpretações
- São baseados em uma relação de confiança

O que é um Contrato Inteligente?

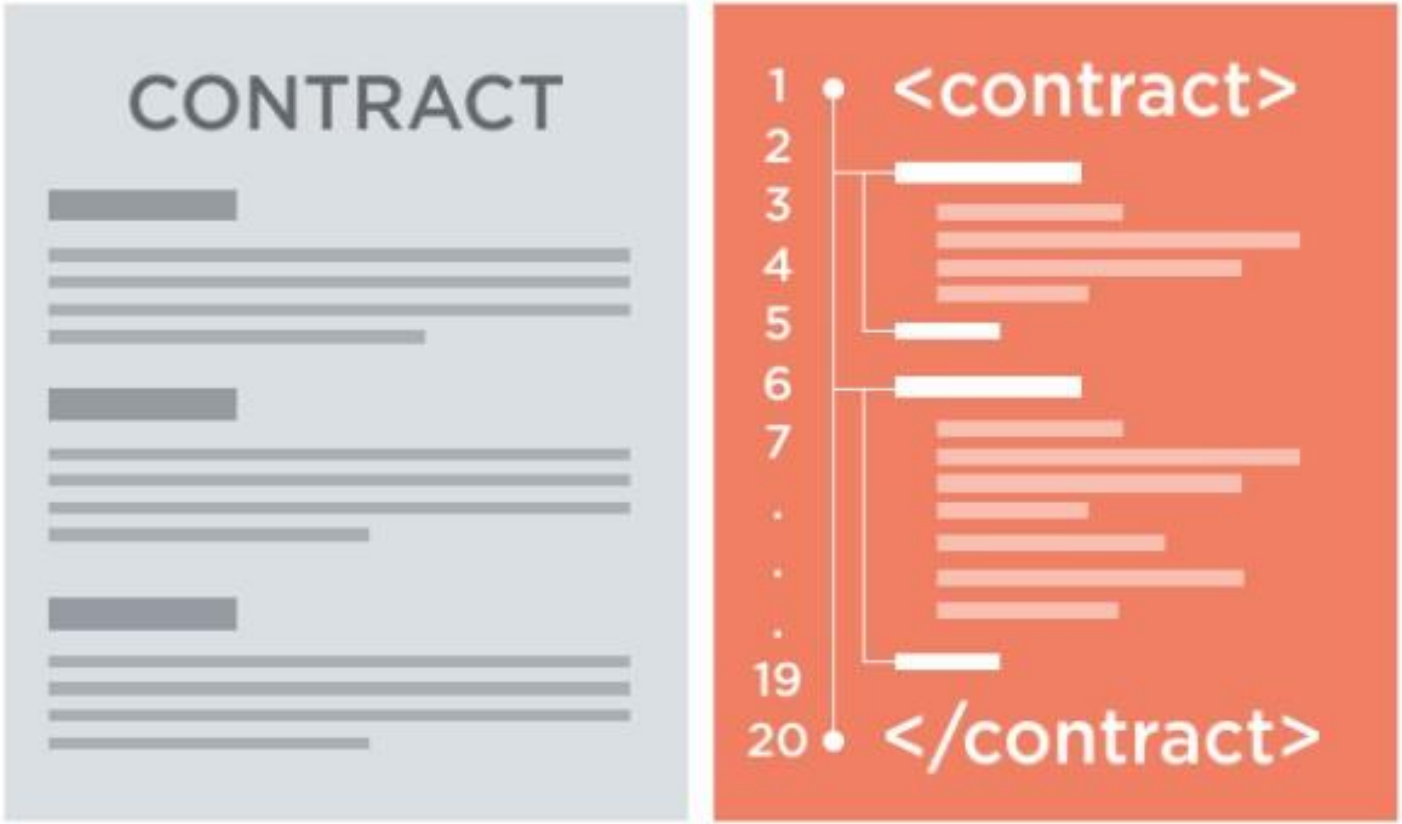
Um conjunto de promessas especificadas em formato digital

Mudança de Paradigma.

Conhecido na literatura como Smart Contract (SC)

confiança de partes não confiáveis acaba sendo exigida

São contratos iguais aos que existem no mundo real só que escritos no formato de um programa de computador cuja a execução impõe os termos do contrato


CONTRACT
1
2
3
4
5
6
7
19
20
<contract>
</contract>

Quais os Objetivos?

Satisfazer Condições Contratuais

Minimizar exceções maliciosas e acidentais

Minimizar a necessidade de intermediários confiáveis

Cenários Problemáticos

E nós Inputamos isso onde?

- Não existe uma garantia da execução dos contratos e do controle dos meios físicos;
- Dificuldade para um único computador garantir a implementação desses termos em um contrato inteligente.

Vamos usar blockchain

Contratos Inteligentes Baseados em Blockchain

A integração da tecnologia blockchain e contratos inteligentes oferece flexibilidade para desenvolver e projetar novos sistemas

Torna-se capaz a resolução de problemas reais sem a necessidade de terceiros de tal forma a garantir menores custos e menores tempos

É possível que os contratos inteligentes sejam executados em código legível por máquina na plataforma blockchain

Processo

Um contato entre as partes é escrito como código na cadeia de blocos. Os indivíduos envolvidos são anônimos, mas o contato é uma razão pública

Um evento é desencadeado como data de caducidade e preço de exercício é atingido e o contrato se executa de acordo com os termos codificados.

Os reguladores podem usar a cadeia de blocos para entender a atividade no mercado, mantendo a privacidade dos envolvidos

Para a criação de um contrato é necessário: o objeto do contrato, assinaturas digitais, termos do contrato e uma plataforma de blockchain descentralizada

Plataformas

Cada plataforma possui suas características, suas particularidades, ambientes de execução e todas rodam em paralelo.

Características

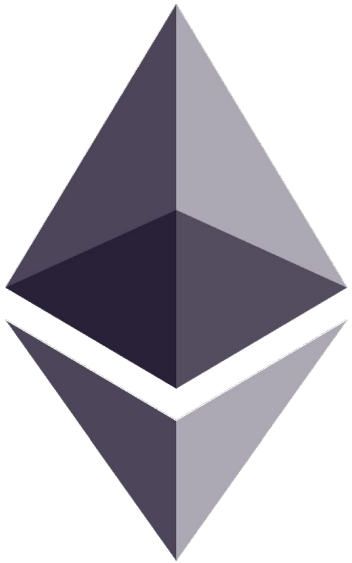
- Públicas
- Privadas
- Escreve Contratos
- Não escreve contratos

ethereum

CARDANO

HYPERLEDGER FABRIC

THE DEVELOPER'S CONFERENCE



ethereum

Por que a Ethereum?

- Pode ser usada para negociar qualquer coisa sem a interferência de uma autoridade central
- É amplamente utilizada para o desenvolvimento de contratos inteligentes
- É amplamente citada e utilizada na literatura acadêmica;

- É uma rede pública
- Possui documentação bem detalhada e uma comunidade ativa
- Possui um número grande e crescente de ferramentas para ajudar os desenvolvedores a criar, testar e implantar seus aplicativos

Linguagens de Programação para a escrita de Contratos Inteligentes para Ethereum

Solidity

THE DEVELOPER'S CONFERENCE

C.E.S.A.R

Orientada a contratos

Baseada em JavaScript

Baseada em JavaScript

Estatisticamente tipada

suporta herança, bibliotecas e tipos complexo

```
contract Puzzle {

  address public owner ;
  bool public locked ;
  uint public reward ;
  bytes32 public diff ;
  bytes public solution ;

  function Puzzle ()  {// constructor
    owner = msg.sender ;
    reward = msg.value ;
    locked = false ;
    diff = bytes32 (11111); // pre-defined
        difficulty
  }

  function (){ // main code , runs at every
      invocation
    if ( msg.sender == owner ) { // update reward
      if ( locked )
        throw ;
      owner.send(reward);
      reward = msg.value ;
    } else if ( msg.data.length > 0) {
      // submit a solution
      if ( locked ) throw ;
      if ( sha256 ( msg.data ) < diff ) {
        msg.sender.send(reward); // send reward
        solution = msg.data ;
        locked = true ;
      }
    }
  }
}
```

Vyper

Orientada a contratos

Baseada em Python

Contratos mais fáceis de entender

Sem grandes verbosidades

Preza Principalmente Simplicidade e Segurança

```
24  def __init__():
25      assert (msg.value % 2) == 0
26      self.value = msg.value / 2  # The seller initializes the contract by
27          # posting a safety deposit of 2*value of the item up for sale.
28      self.seller = msg.sender
29      self.unlocked = True
30
31  @public
32  def abort():
33      assert self.unlocked #Is the contract still refundable?
34      assert msg.sender == self.seller # Only the seller can refund
35          # his deposit before any buyer purchases the item.
36      selfdestruct(self.seller) # Refunds the seller and deletes the contract.
37
38  @public
39  @payable
40  def purchase():
41      assert self.unlocked # Is the contract still open (is the item still up
42                           # for sale)?
43      assert msg.value == (2 * self.value) # Is the deposit the correct value?
44      self.buyer = msg.sender
45      self.unlocked = False
46
47  @public
48  def received():
49      # 1. Conditions
50      assert not self.unlocked # Is the item already purchased and pending
51                               # confirmation from the buyer?
52      assert msg.sender == self.buyer
53      assert not self.ended
54
55      # 2. Effects
56      self.ended = True
57
58      # 3. Interaction
59      send(self.buyer, self.value) # Return the buyer's deposit (=value) to the buyer.
60      selfdestruct(self.seller) # Return the seller's deposit (=2*value) and the
```

Resumo

Platform	Ledger, Consensus	OP Codes / Language	Features
Bitcoin	UTXO, PoW	Script† / Ivy	Linear execution conditions, no loops
Ethereum	Accounts, PoW→PoS	EVM / Solidity	General purpose computing
Neo	Accounts, D-BFT	NeoVM / C#, Java, ...	Many compilers for high-level languages
NXT	Accounts, PoS	Templates† / Website Forms	Just parameters, no coding
Corda	UTXO, Raft	JVM / Java, Kotlin	stateless functions, links legal prose
Cardano	UTXO, PoS	IELE / Plutus	functional programming
Tezos	Accounts, PoS	Michelson / Liquidity	formal verification

Table I: Different platforms that implement smart contracts.
†: language limited and *not* Turing-complete.

C.E.S.A.R

Desafios e Desvantagens dos Contratos Inteligentes Baseados em Blockchain

Desafios

THE DEVELOPER'S CONFERENCE

C.E.S.A.R

Escalabilidade da execução dos contratos

Flexibilidade e problemas de privacidade

Escalabilidade da execução dos contratos

Dependência do timestamp da máquina mineira

Dependência de ordenação de transações

Testar os Contratos

Imutabilidade

Dificuldades na escrita de Contratos Inteligentes

Dificuldades para a escrita dos contratos

Uma questão crucial durante o desenvolvimento de contratos inteligentes, baseados em blockchain, é permitir que especialistas de domínio compreendam, discutam e especifiquem o contrato de forma colaborativa

(HE et al., 2018)

A complexidade da codificação de contratos inteligentes e a necessidade de fazê-lo corretamente limitam a adoção e a aceitação dessa tecnologia

(FRANTZ; NOWOSTAWSKI,2016)

o principal desafio na escrita de contratos inteligentes é desenvolver/utilizar uma linguagem de especificação de contrato inteligente que seja legível por humanos, compilável para código e legalmente executável

(Regnath e Steinhorst, 2018)

E agora? o que nós faz?

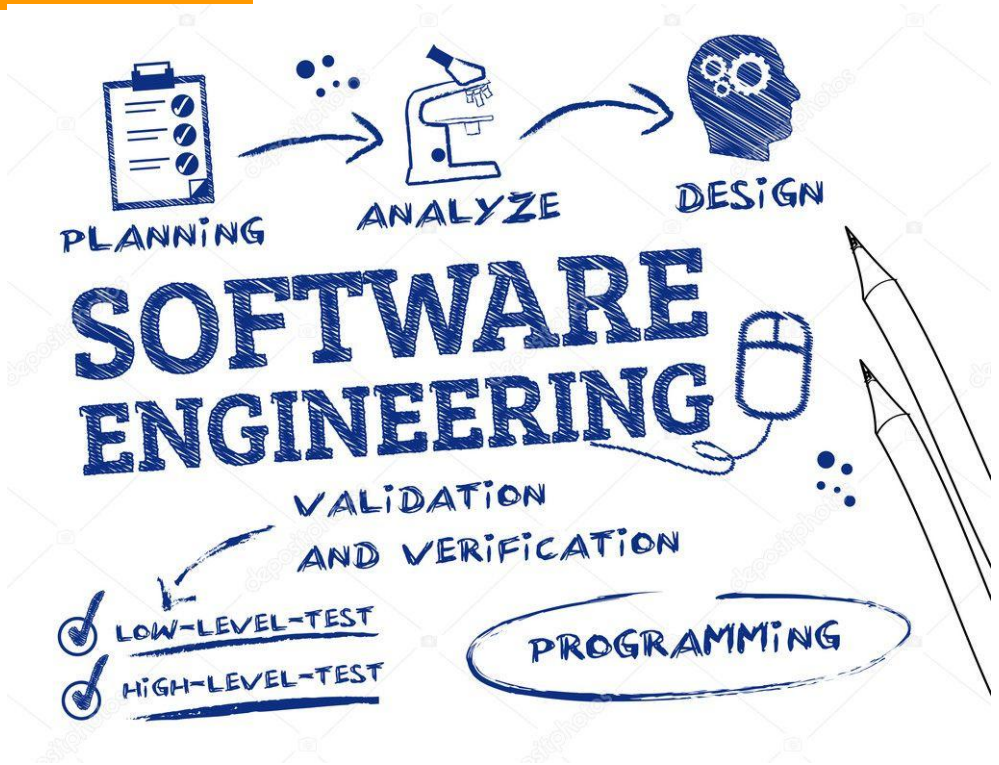
Engenharia de Software

ES vem aqui me socorrer

- Como forma de contribuição para a melhoria da facilidade da escrita dos contratos vamos fazer uso de 2 conceitos da Engenharia de Software

Conceitos Base

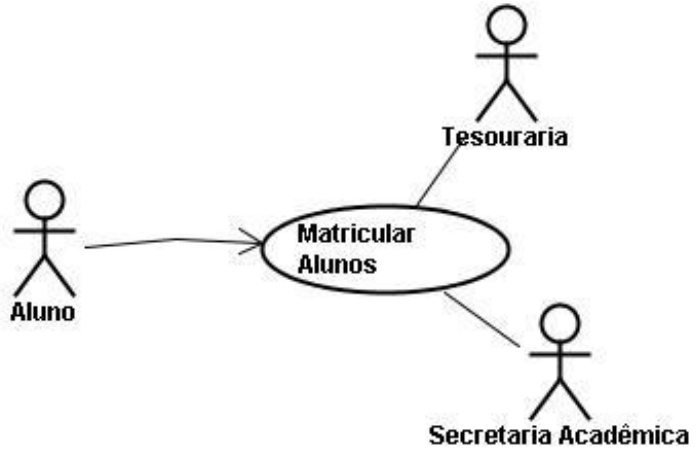
- Desenvolvimento Dirigido Por Modelos (MDE)
- Linguagens de Modelagem Específica de Domínio (DSML)



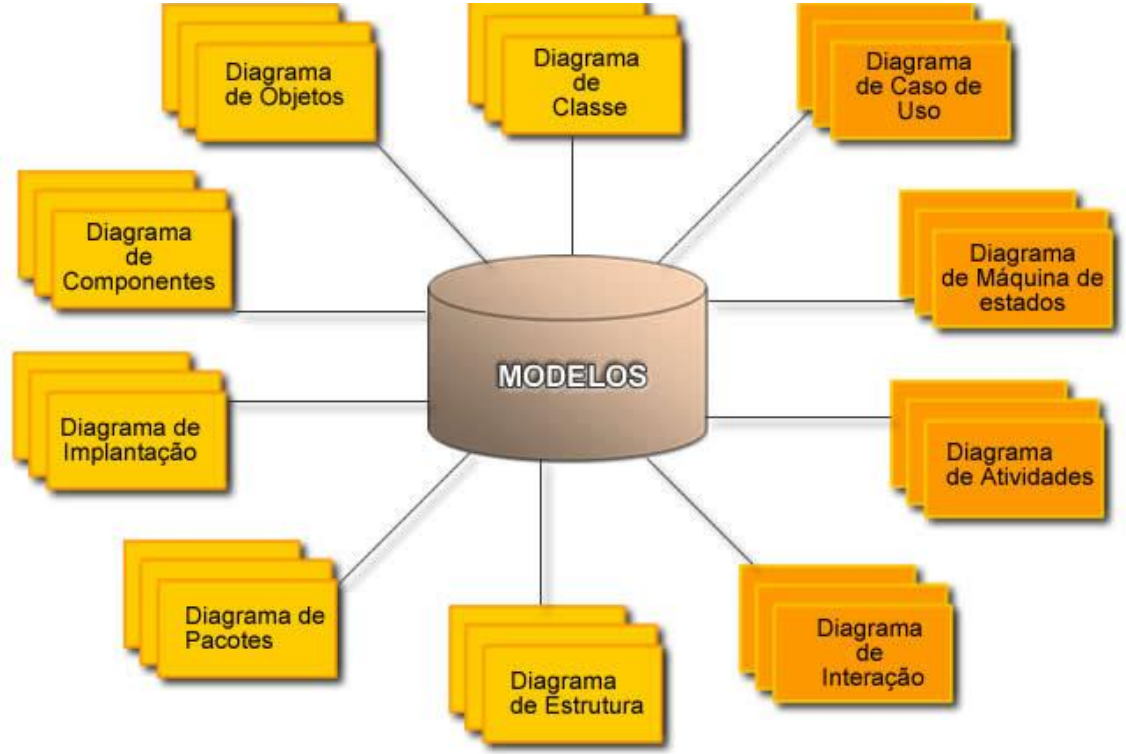
Desenvolvimento Dirigido Por Modelos (MDE)

Como funciona o versionamento de dados?

- Um paradigma na engenharia de software que faz uso de modelos para testar, simular, analisar e manter os sistemas a serem construídos, entre outras atividades
- O paradigma MDE preconiza que o desenvolvimento inicial e modificações futuras da aplicação sejam efetuados apenas no modelo mais abstrato
- Geração automática de Código



É tudo modelo



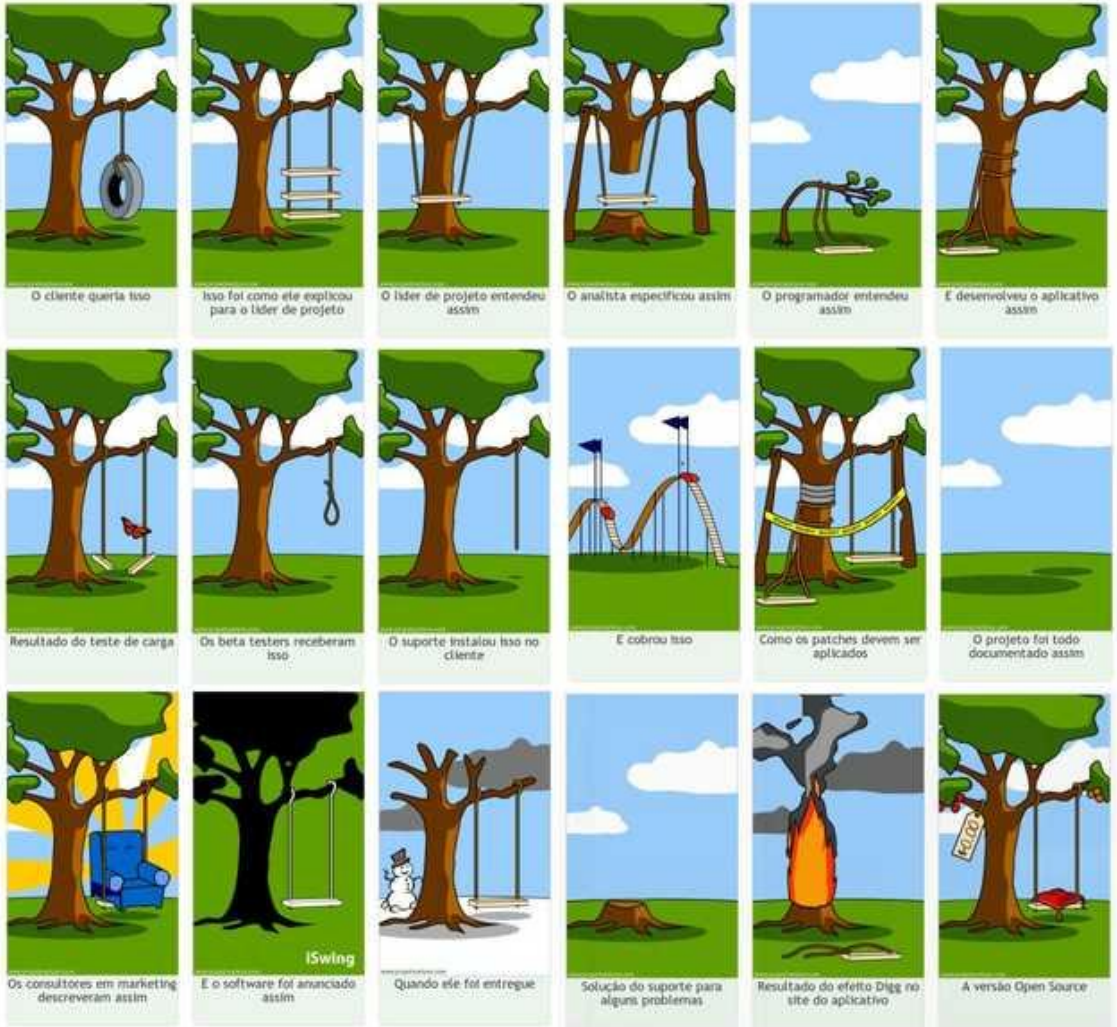
O Desenvolvimento dirigido por modelos é baseada em dois princípios

Linguagens de Modelagem Específicas de Domínio

Desenvolvida para atender um determinado domínio de aplicação.

Mecanismos para Transformações

A partir de modelos é permitido gerar os artefatos necessários para a implementação da aplicação



Aquela velha história...

C.E.S.A.R

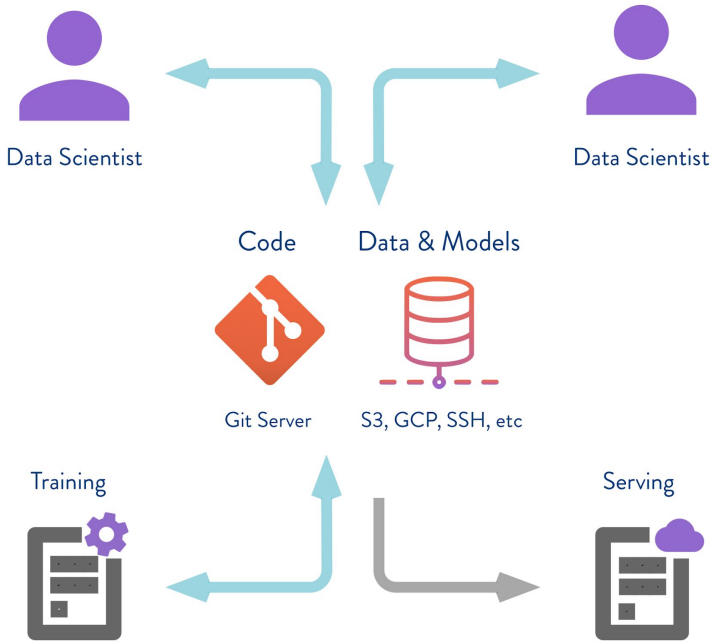
Linguagens de Modelagem Específica de Domínio (DSMLS)

DSMLs

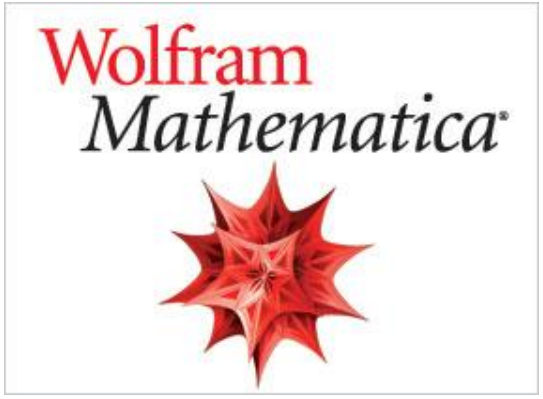
Linguagens de Modelagem específica de Domínio ou Domain-specific modeling languages (DSMLs)

- São linguagens de modelagens especializadas para um domínio de aplicação específico.
- As DSMLs aumentam o nível de abstração e, ao mesmo tempo, mantêm o espaço de design limitado a um domínio específico

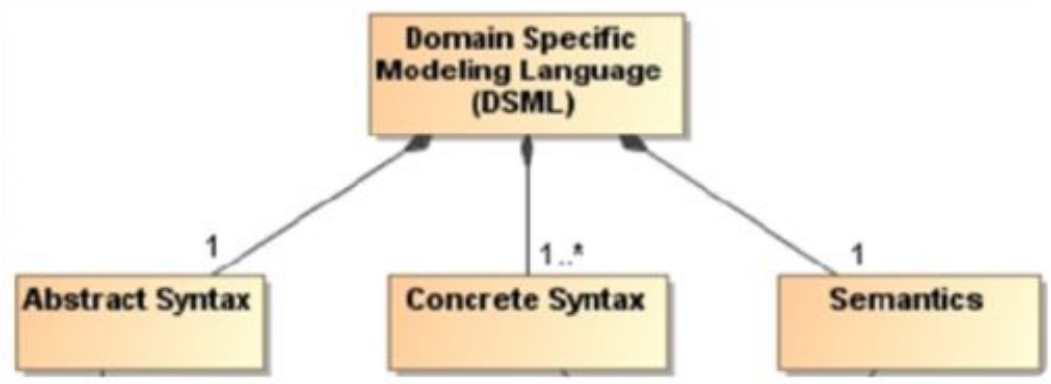
- Podem ser visuais ou textuais



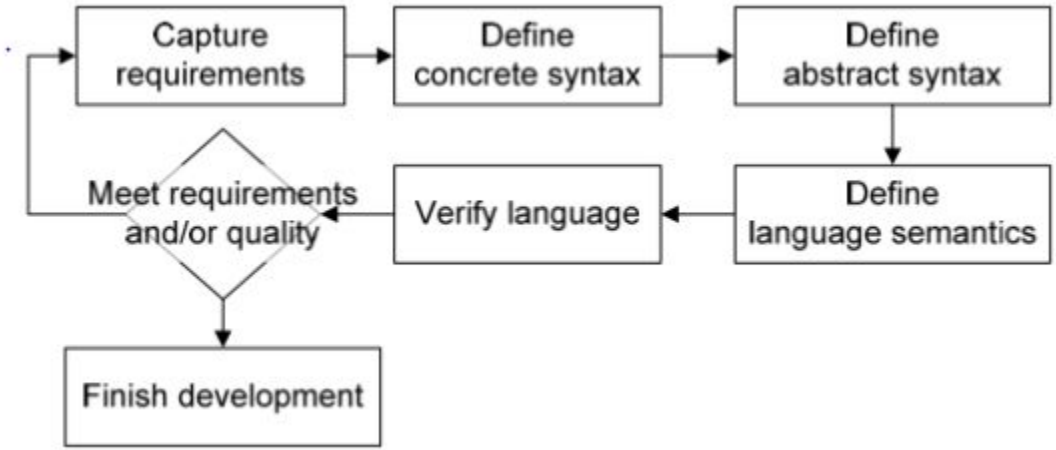
Exemplos



A Composição



Processo de desenvolvimento de uma DSML



C.E.S.A.R

DSMLs Desenvolvidas para a Escrita de Contratos Inteligentes Baseados em Blockchain

Linguagem SPESC

```
contract Purchase {
  party Seller {/* details of Seller */}
  party Buyer {/* details of Buyer */}
  info : ProductInfo // contract property
  term No1 : … /* details of Term */
  term No2 : …
  …/* other Terms */
  type ProductInfo {
    price : Money,
    model :  String
  }
}
```

```
Contract in SmaCoNat version 0.1.

§ Involved Accounts:
Account 'BarrierIn' by 'AComp' by Genesis alias 'BarrierIn'.
Account 'BarrierOut' by 'AComp' by Genesis alias 'BarrierOut'.

§ Involved Assets:
Asset 'TheCoin' by Genesis alias 'TheCoin'.
Asset 'ParkTicket' by Self alias 'Ticket'.
Asset 'OpenBarrier' by Self alias 'Open'.

§ Agreement:
Self issues 'Ticket' with value 42.
Self issues 'Open' with value 1.

§ Input Event:
if Input is equal to 'TheCoin' from Anyone
and if value of Input is equal to 0.3
then
    Self transfers 'Ticket' with value 1 to owner of Input.
    Self transfers 'Open' with value 1 to 'BarrierIn'.
    Self issues 'Open' with value 1.
endif

if Input is equal to 'Ticket' from Anyone then
    Self transfers 'Open' with value 1 to 'BarrierOut'.
    Self issues 'Open' with value 1.
endif
```

Linguagem SmaCoNat

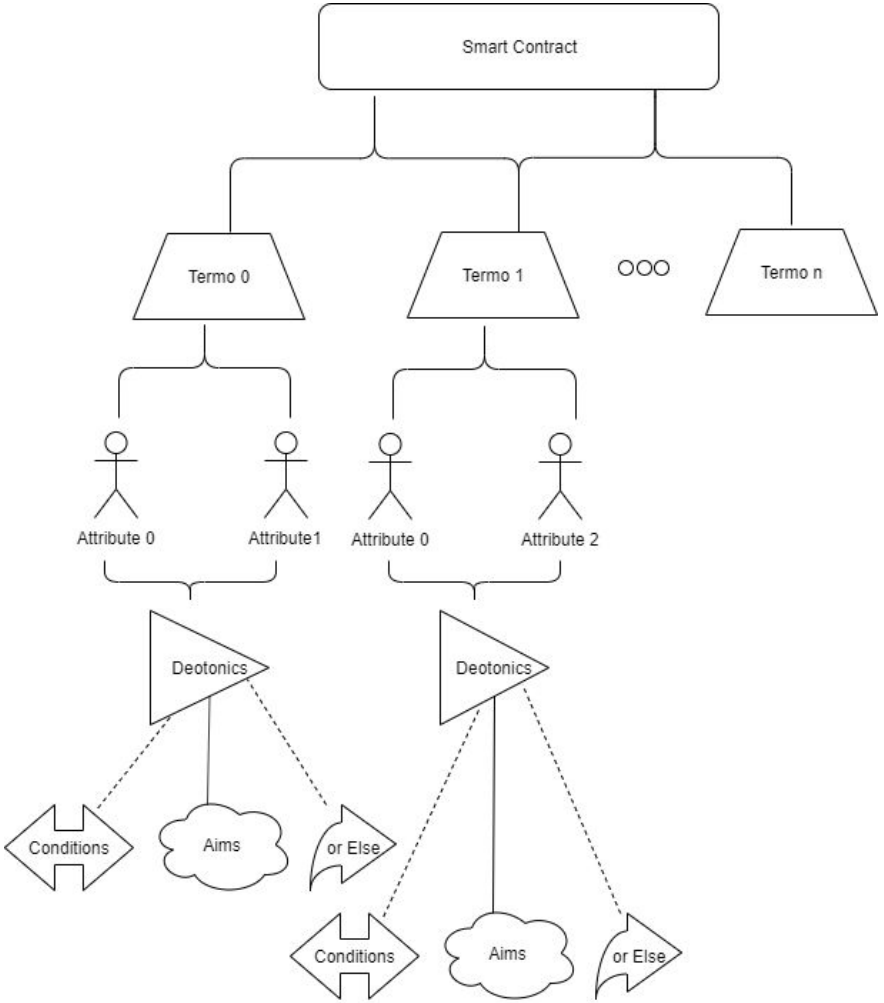
```
Adico(
  A("buyer"),
  D(must),
  I("pay", object("funds")),
  C("before", "deadline"),
  O(Adico(
      A("system"),
      D(must),
      I("release", object("objectOfInterest"),
        target("seller", "address")))
   AND
    Adico(
      A("system"),
      D(must),
      I("send", object("funds"),
        target("buyer", "address")))
      )
) AND
  Adico(
    A("system"),
    D(must),
    I("send", object("funds"),
      target("seller", "address")),
    C(IF("msg.value", Operator.>=, "price"))) AND
  Adico(
    A("system"),
    D(must),
    I("release", object("objectOfInterest"),
      target("buyer", "address")),
    C(IF("msg.value", Operator.>=, "price")))
```

Linguagem ADICO

- Atributos - descrevem as características ou atributos de um ator.
- Deontic - descreve a natureza da declaração como uma obrigação, permissão ou proibição.
- AIm - descreve a ação ou resultado que esta declaração regula.
- Condições - descreve as condições contextuais sob as quais esta declaração é válida. Se não for especificado, a declaração institucional é válida sob qualquer circunstância.
- Ou - descreve as consequências associadas à não conformidade

Minha Contribuição

- Desenvolvimento de uma DSML visual para a criação de contratos inteligentes para a plataforma Ethereum.
- Contribuição para a melhoria da facilidade da escrita dos contratos.
- Geração de códigos de contratos inteligentes de forma semi-automática.



THE DEVELOPER'S CONFERENCE

C.E.S.A.R

SMaCoViL
Uma Linguagem visual

O Processo...


Linguagem Visual SMaCoVil
Contrato em Adico
Contrato em Solidity
Deploy do Contrato na Ethereum

Próximos Passos

- Terminar a Implementação da Linguagem;
- Realizar experimentos;
- Validar Linguagem;
- Avaliação e Melhorias;

Experimentos

- Grupo de pessoas (dentro e fora do setor de TI) vão utilizar a linguagem para gerar contratos e comprovar o quão a SMaCoVil melhorou ou não a facilidade na escrita de um contrato inteligente

Validação

- Validar através de métricas de linguagens/DSMLS dentro da literatura científica

  ou

- Grupo Focal

THE DEVELOPER'S CONFERENCE

C.E.S.A.R

Perguntas

Obrigada!