Towards high-level specification of Smart-Contracts

Juan Manuel Vara

1: Kybele Research Group | Universidad Rey Juan Carlos

@jmvara | @KybeleResearch | @ISE_URJC

Joint work with
Cristian Gómez[1], Francisco J. Pérez[1], E. Marcos[1]

Agenda

1. Blockchain
2. Smart Contracts
3. Bringing models to the rescue
4. The road ahead

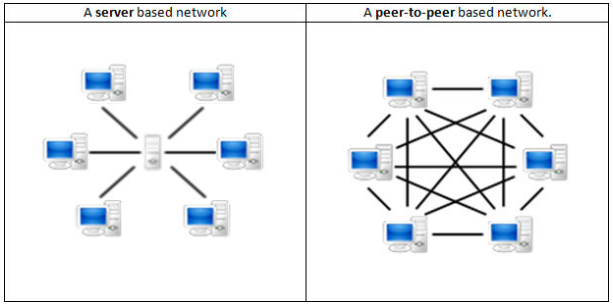

THE BIGGEST INVENTION SINCE THE EMERGENCE OF THE INTERNET!!
https://www.healthnewsreview.org/wp-content/uploads/2016/12/iStock-487078483.jpg

Blockchain

- What is a Blockchain?
  - A distributed DB + Encryption + Immutability + stored procedures (smart contracts)
    - A blockchain is a list (chain) of groups (blocks) of transactions
    - Like traditional DDBBs they can be used for anything a DB is used.
- How does it work?
  - Interested subjects add transactions to the pool
  - Nodes verify and add them to some block on the ledger
  - Ledger is replicated among distributed nodes
  - Eventual consistency
    - In the absence of centralized control, all nodes eventually achieve consensus about the content of the ledger
  - Append-only data structure
    - May add transactions – Nearly imposible to change data

Blockchain

- Why so much hype?
  - New businesses and business models are expected to arise, but **as yet** there are not a lot of examples of significant use in production of blockchain systems within industry or government.
- <u>Disintermediation</u>
  - Direct VS Indirect interactions: less processing time and lower costs
- (Improved) Peer-to-peer systems
  - E.g.: music → inmaterial nature / low costs of data transfer



https://dollarsandsensemagazine.com/wp-content/uploads/2015/02/Depositphotos_5999243_s-e1425140615493.jpg

Blockchain

A tool for achieving and maintaining integrity in purely distributed peer-to-peer systems …

… that consists of an unknown number of peers with unknown reliability and trustworthiness.

Ownership and Witnesses

- Having one witness is good, but having many independent witnesses is the key.
- Instead of one ledger, a p2p system of ledgers …

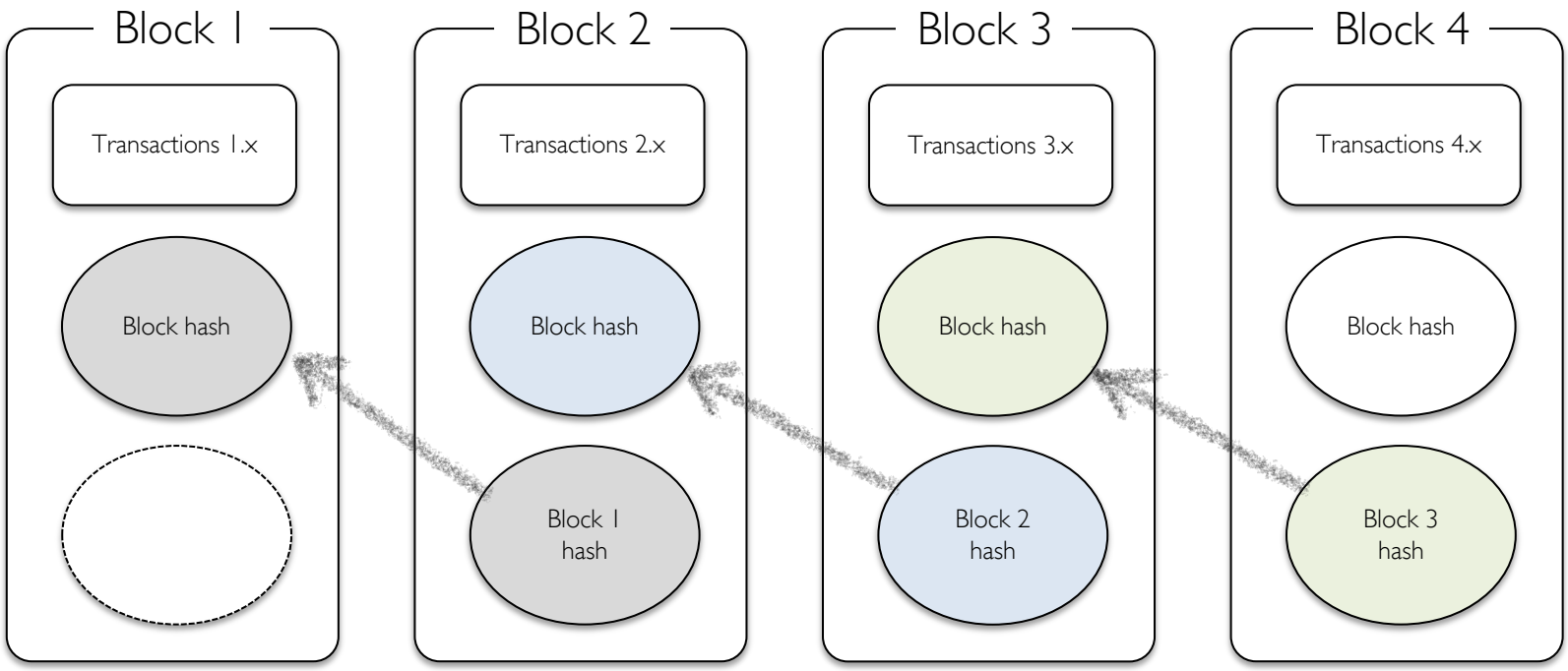
Blockchain Definition

The blockchain is a purely distributed peer-to-peer system of ledgers that utilizes a software unit that consist of an **algorithm**, which negotiates the informational content of ordered and connected blocks of data together with cryptographic and security technologies in order to achieve and maintain its integrity.

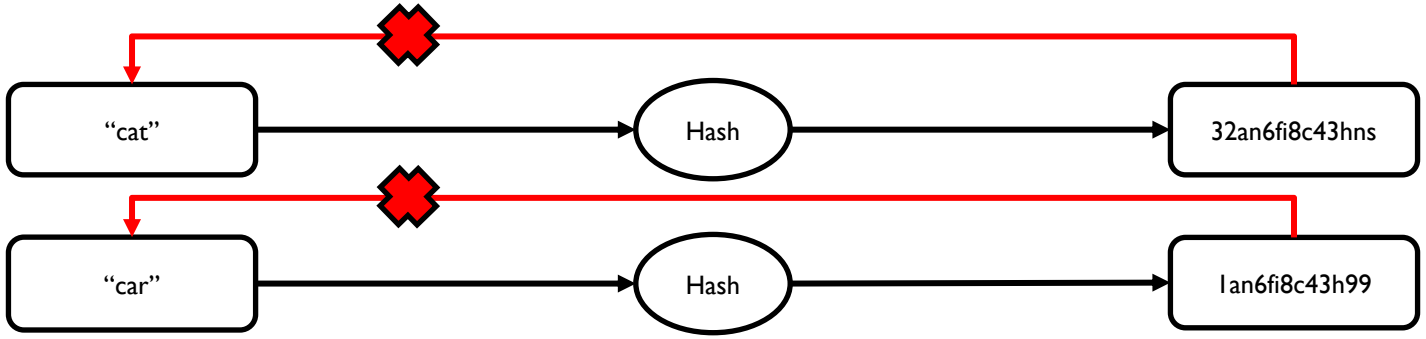
Drescher, D. *Blockchain Basics* (2017).

THE Block-Chain ☺

- General view of the blockchain



Hashing



Hash functions are small computer programs that transform any kind of data into a number of fixed length (Bitcoin – sha256), regardless of the size of the input data.

- Hashing in the blockchain
  - Storing data in a change-sensitive manner
  - Incur computational costs for changing the data-structure

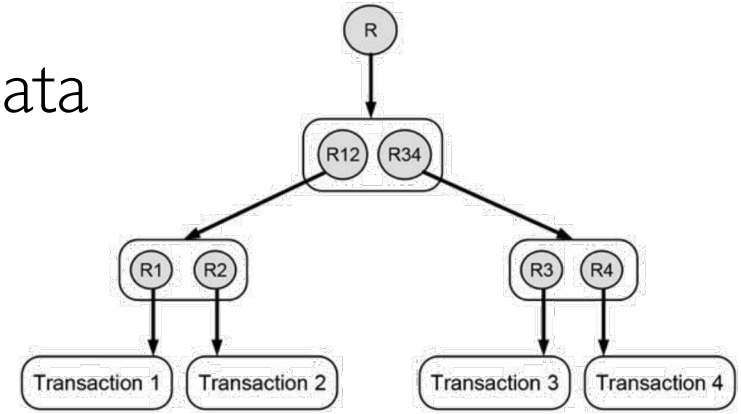
Storing data in a change-sensitive manner

- Hash references are used to create an **ordered** list of blocks with hash references to previous block

(Drescher, 2017)

- *Knitting* :)
  - Adding a new block at the end of the blockchain-data-structure is easy, while changing data located somewhere in the chain is quite elaborate

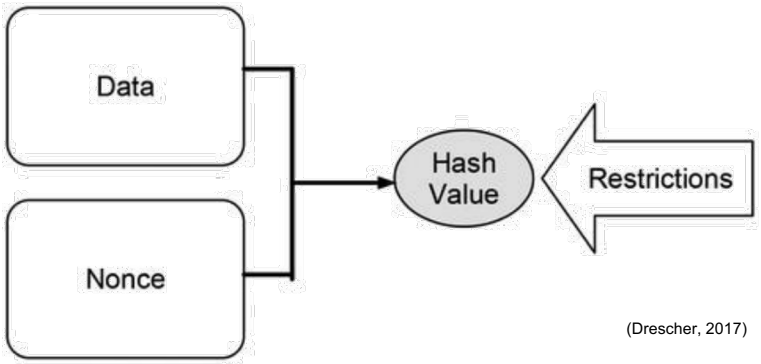
- Each block stores the data into a *Merkle* tree



(Drescher, 2017)

Causing Time-Consuming Computations

- Hash puzzles
  - Akin to **combination locks** to be openned by trial an error:
    - A specific lock that requires a unique sequence of numbers in order for it to be opened.
    - Opening is based on sheer diligence and hard work.



(Drescher, 2017)

Nonce	Text to Be Hashed	Output
0	Hello World! 0	4EE4B774
1	Hello World! 1	3345B9A3
2	Hello World! 2	72040842
3	Hello World! 3	02307D5F
	…	
613	Hello World! 613	E861901E
614	Hello World! 614	**00068A3C**
615	Hello World ! 615	5EB7483F

What data combined with Hello World! would yield a shortened hash value with **three leading zeros**?

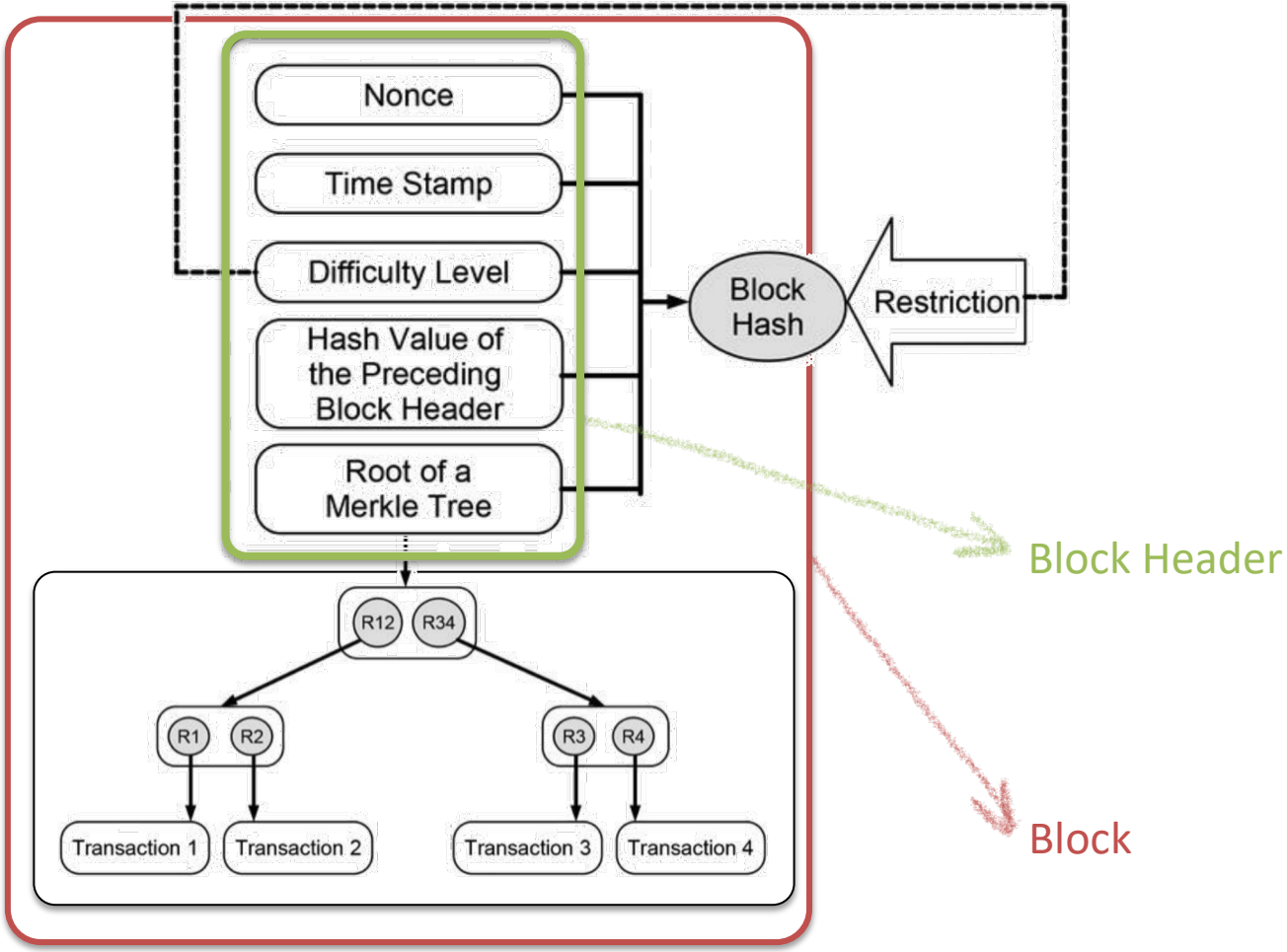
Difficulty

THE PUZZLE

Find the nonce that combined with Hello World! yields a shortened hash value that starts with three leading zeros

Adding a block to the Blockchain

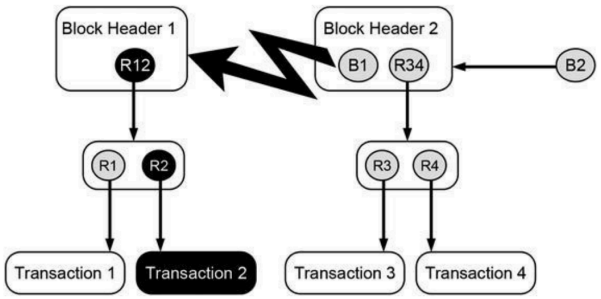
- Making Adding Data Computationally Expensive

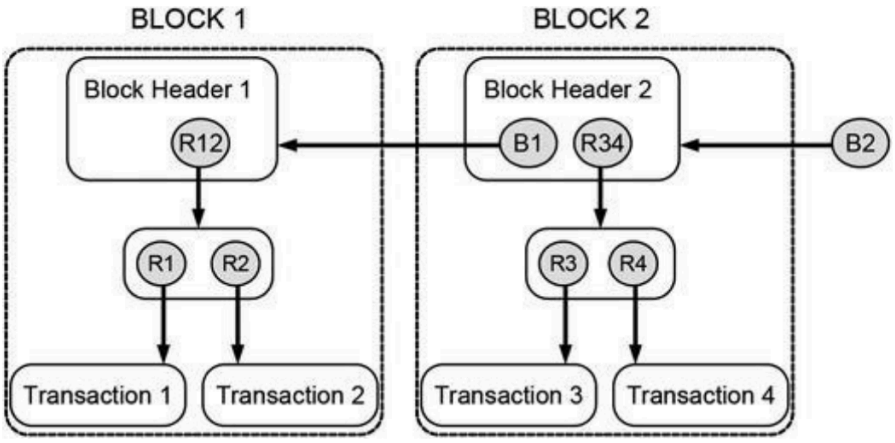


(Drescher, 2017)

Blockchain Inmutability

- Inmutability
  - Making manipulations stand-out
  - Enforcing Rewriting the History for Embedding Changes
  - Making Adding Data Computationally Expensive





(Drescher, 2017)

Manipulate existing T_x

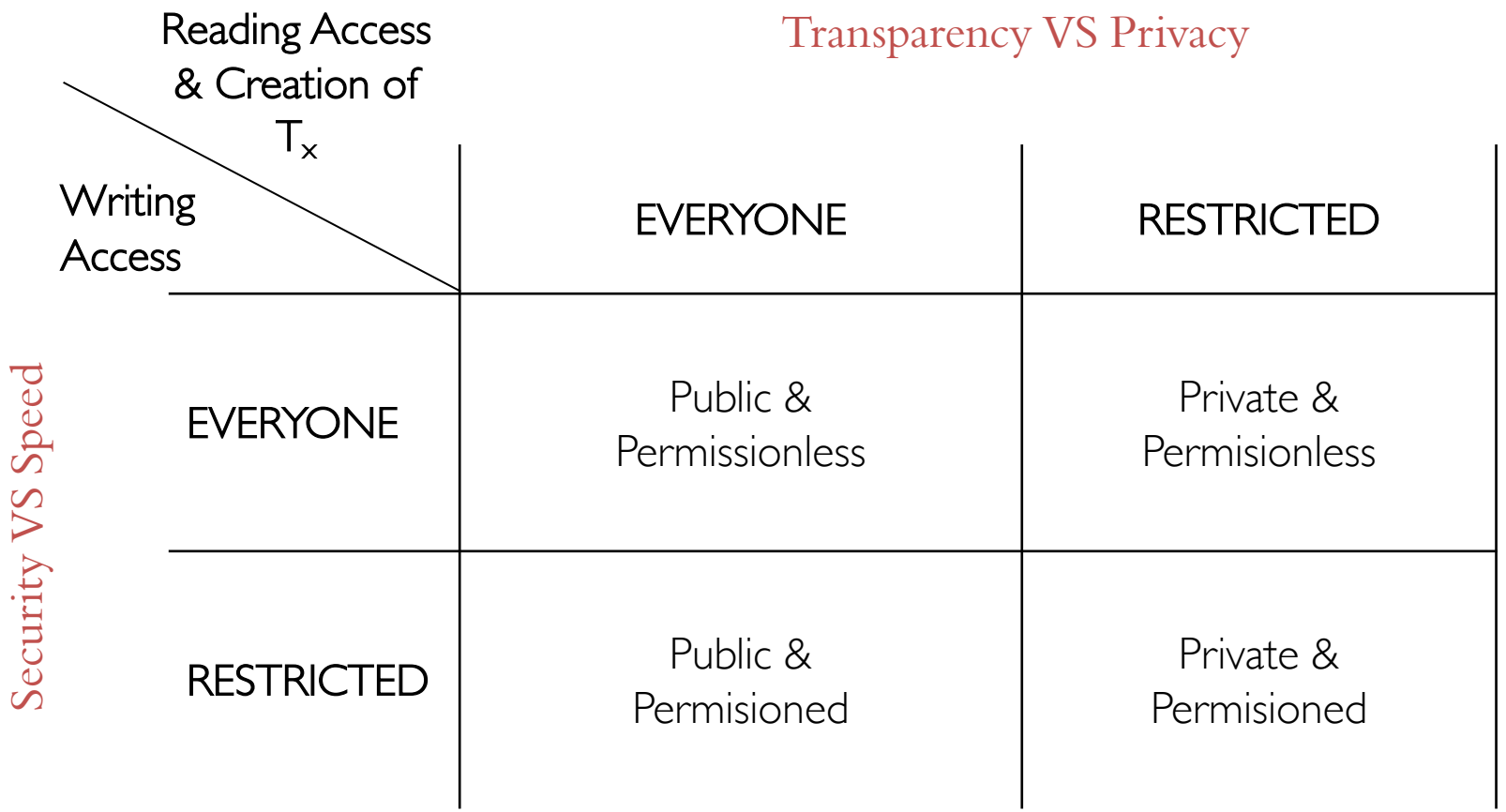
If solving a hash puzzle takes on average 10 minutes, 210 minutes are needed to embed a manipulation in a T_x that belongs to a block header located 20 blocks below the current head.

1. Rewrite the Merkle tree of manipulated T_x
2. Rewrite block header of the rewritten tree
3. Rewrite all succeeding block headers up to the head of the blockchain data-structure

Blockchain – Rewarding System

- Blockchain-algorithm allow all nodes of the system
  - To act as **supervisors** of their peers
  - **Reward/punish** them for adding valid and authorized transactions
- Every node is in one of two-steps algorithm
  - Evaluating a new block that was created by others
    - If a block is removed from the blockchain-data-structure, then the reward for adding it is withdrawn from the node that initially received it.
  - Trying hard to be the next node that creates a new block that has to be evaluated by all others
    - The node whose block was accepted will receive the **fees** for all transactions contained in the block as reward.

Blockchain Versions



Blockchain Versions

- Public
  - Any one can become a public node in the chain, which allow them to perform, validate and view transactions in that network.
- Semi-public or federated (Consortium):
  - The nodes must be identified before they can interact on that network (Private & Permissioned).
  - Recommended for private companies or governments that want transparency in their actions.
- Private (federated + central control):
  - Preset private nodes with privileges.
  - Federated ones, but an entity is in charge of controlling everything.

Most Popular Blockchains (or so …)

- Features of the Bitcoin network
  - Focused on providing an alternative to conventional currencies
    - The cryptocurrency that operates on that network is Bitcoin(deflationary).
  - Rewards for block validation

BLOCKCHAIN AS A DATA STORE

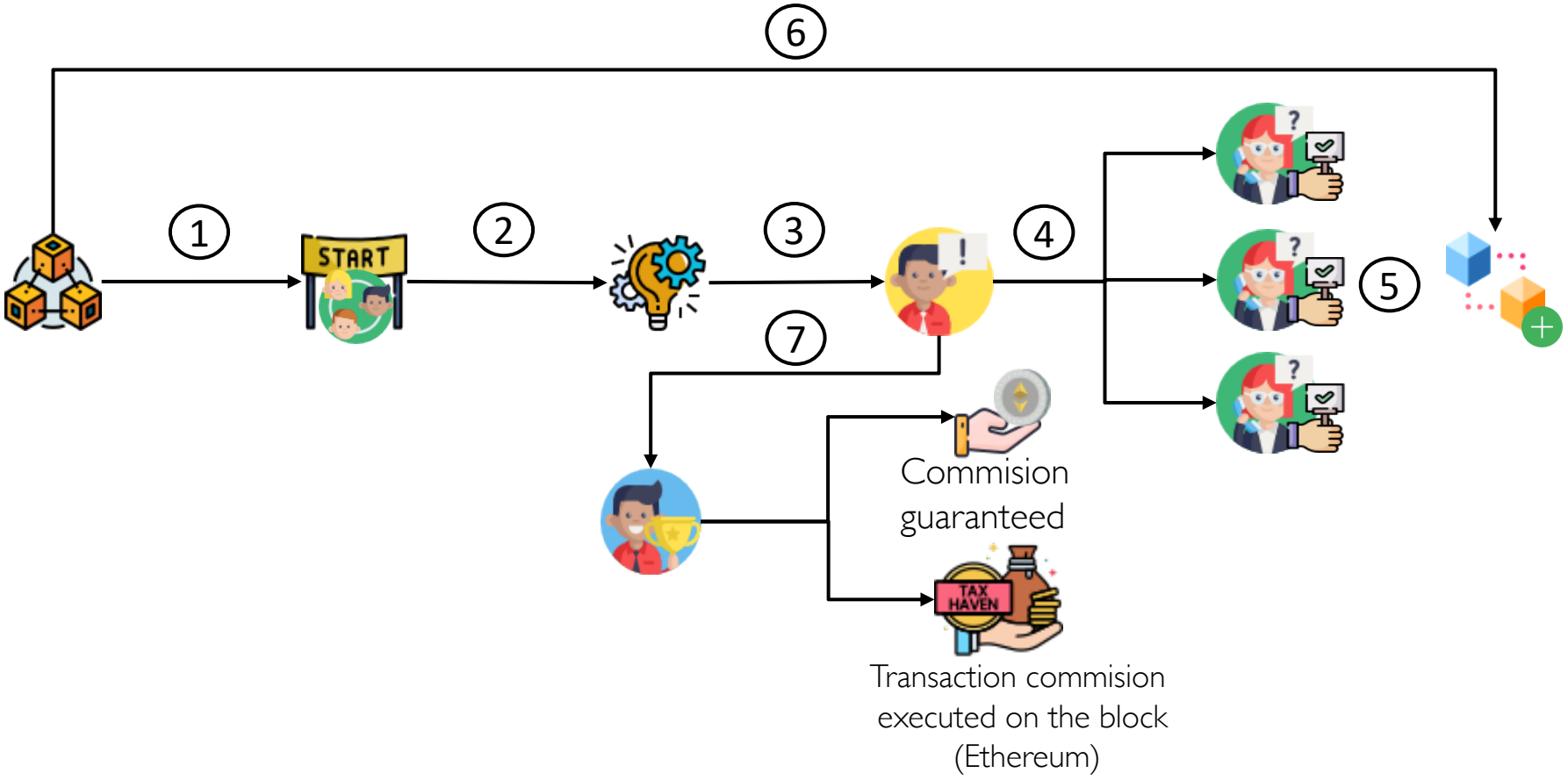
ethereum

- Features of the Ethereum network
  - Focused on Smart Contracts and dapp execution (EVM)
    - The cryptocurrency that operates on that network is Ether (inflationary)
      - Merely a way to facilitate and monetize the opetarion of Ethereum
    - Rewards for block validation, transaction validation and Smart Contracts execution.

BLOCKCHAIN AS A COMPUTATIONAL INFRASTRUCTURE

Block Generation / Consensus

- Block generation
  - Proof of Work
  - Proof of stake



Block Generation / Consensus

- Block generation

Proof of work

Proof of Stake

- The probability of mining a block is proportional to the amount of cryptocurrency available to the miner.
  - Highest priority for users who have more currency because they are the most interested in maintaining the ecosystem correctly

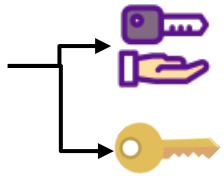
- Advantages:
  - Energy saving
    - Computational power is less necessary than in PoW in order to validate a block.

Blockchain Actors

- Developers
  - Implement the protocol.

- Users

  - Connect to the network to perform transactions
  - Wallet



- Miners

  - Responsible for validating the blocks where transactions are recorded in exchange of rewards.
  - Pools: Set of miners that come together to mine blocks in a blockchain network

Success Stories

- Academic credentials:
  - MIT, Cyprus

- Medical data register
  - MedRec.

- Supply chains:
  - Carrefour, Nestle.

- Sports betting exchange and lottery applications:
  - Peerplays, Wagger

- Secure digital identity solutions:
  - Illinois state

Agenda

1. Blockchain
2. Smart Contracts
3. Bringing models to the rescue
4. The road ahead

Smart Contracts

- Computer programs
  - Hosted on Ethereum
  - Executes **autonomously** the clauses collected in it when the conditions are satisfied
    - DTL as a DDBB
    - Smart Contracts as triggers or microservices where the business logic transacting with that data lives
  - Blockchain technology "Sets in stone" the agreement
    - The contract inherits trust-less, immutability, transparency …

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

1. Conditions are programmed
2. Implied parties sign the conditions (program)
3. Contract is *placed* into a blockchain so no one could modify it

VS CONVENTIONAL CONTRACTS

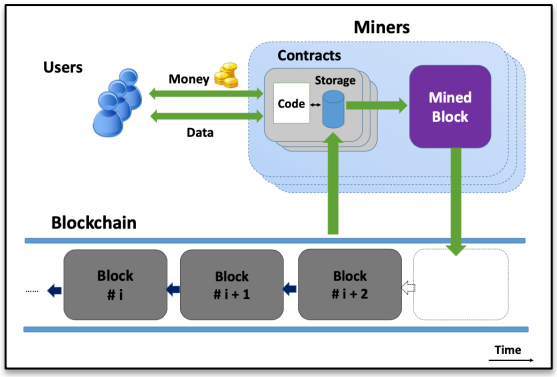
- A program does not leave space to different interpretations: disambiguation
- No need of a trusted third-party → ↯Transaction Costs
- Time-efficient
- Data Storage (future disputes)

SOLIDITY

How does a Smart Contract work?

An instance of program code that runs in the blockchain

Program code | Storage file | Account balance



(Delmolino et al., 2016)

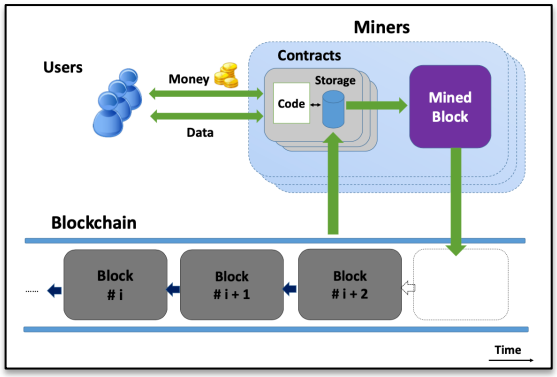
SMART CONTRACT MODEL

1. User create the contract: transaction posting
   a) Code cannot be changed
   b) Storage file stored in the blockchain
2. Contract is executed upon message received (either users or contracts)
   a. Read/write from its file
   b. Recieve / Send money from its account balance from/to users (contracts) invoke the contract
3. Miners reach consensus on the output of the execution and update the blockchain accordingly

How does a Smart Contract work?

An instance of program code that runs in the blockchain

Program code | Storage file | Account balance



(Delmolino et al., 2016)

CONTRACT INVOCATION – T_X AS FUNCTION CALLS

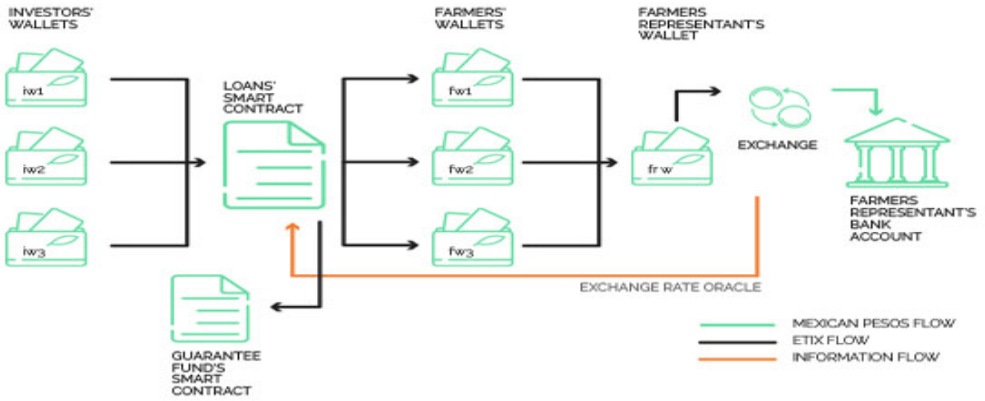
- Contract code will be invoked whenever it receives a T_x from a user
- Multiple entry points of execution – each one is defined as a function
  - After processing the message, contract can return value back to the sender
- The content of the T_x will specify the entry point at which the contract's will be invoked

GAS – DISCOURAGING OVER CONSUMPTION OR RESOURCES

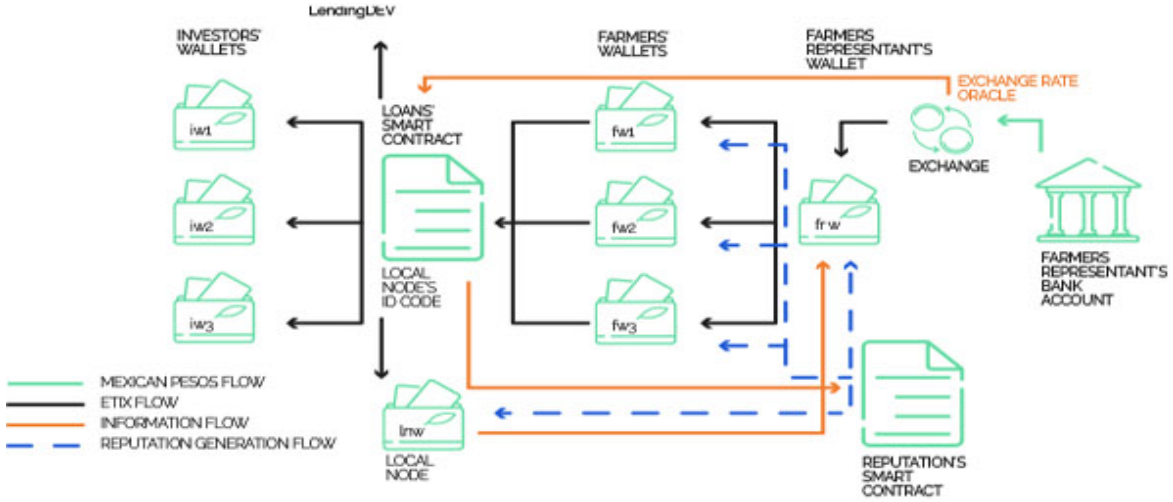
- The user who creates a T_x must spend currency to purchase gas
- Each program instruction consumes gas
- If gas runs out before T_x reaching an ordinary point, exception raised

Smart Contracts-based crowdlending

- Investment



- Payment



https://www.crowdlending.es/blog/que-es-ethic-hub

Programming Smart Contracts

- Libraries & Interfaces
- Global variables
- Events / Modifiers
- Contract signature (*is*)
- Constructor
- Functions

Similar to a Class in any OOPL

SOLIDITY

```
/// Create a simple auction with `_biddingTime`
/// seconds bidding time on behalf of the
/// beneficiary address `_beneficiary`.
constructor(
    uint _biddingTime,
    address payable _beneficiary
) public {
    beneficiary = _beneficiary;
    auctionEndTime = now + _biddingTime;
}

/// Withdraw a bid that was overbid.
function withdraw() public returns (bool) {
    uint amount = pendingReturns[msg.sender];
    if (amount > 0) {
        // It is important to set this to zero because the recipient
        // can call this function again as part of the receiving call
        // before `send` returns.
        pendingReturns[msg.sender] = 0;

        if (!msg.sender.send(amount)) {
            // No need to call throw here, just reset the amount owing
            pendingReturns[msg.sender] = amount;
            return false;
        }
    }
    return true;
}
```

```
# Create a blinded auction with `_biddingTime` seconds bidding time and
# `_revealTime` seconds reveal time on behalf of the beneficiary address
# `_beneficiary`.
@public
def __init__(_beneficiary: address, _biddingTime: timedelta, _revealTime: timedelta):
    self.beneficiary = _beneficiary
    self.biddingEnd = block.timestamp + _biddingTime
    self.revealEnd = self.biddingEnd + _revealTime


# Withdraw a bid that was overbid.
@public
def withdraw():
    # Check that there is an allowed pending return.
    pendingAmount: wei_value = self.pendingReturns[msg.sender]
    if (pendingAmount > 0):
        # If so, set pending returns to zero to prevent recipient from calling
        # this function again as part of the receiving call before `transfer`
        # returns (see the remark above about conditions -> effects ->
        # interaction).
        self.pendingReturns[msg.sender] = 0

        # Then send return
        send(msg.sender, pendingAmount)
```

Programming Smart Contracts

```
1  pragma solidity >=0.4.22 <0.6.0;
2
3  contract Purchase{
4
5      uint public val;
6      address payable public seller;
7      address payable public buyer;
8
9      enum State{Created,Locked,Inactive};
10     State public state;
11
12     constructor()public{
13         seller = msg.sender;
14         val = msg.value /2;
15         require((2 * val) == msg.value,"Value has to be even.");
16     }
17
18     modifier condition(bool condition){
19         require(condition);
20         _;
21     }
22
23     modifier onlyBuyer(){
24         require(msg.sender == buyer,"Only buyer can call this.");
25         _;
26     }
27
28     modifier onlySeller(){
29         require(msg.sender == seller,"Only seller can call this.");
30         _;
31     }
32
33     modifier inState(State _state){
34         require(state == _state);
35         _;
36     }
37
38     event Aborted();
39     event PurchaseConfirmed();
40     event ItemReceived();
41
42     function abort()public onlySeller inState(State.Created){
43         emit Aborted();
44         state = State.Inactive;
45         seller.transfer(address(this).balance);
46     }
47
48     function confirmPurchase() public inState(State.Created) condition(msg.value == (2 * val)) {
49         emit PurchaseConfirmed();
50         buyer = msg.sender;
51         state = State.Locked;
52     }
53
54     function confirmReceived() public onlyBuyer inState(State.Locked){
55         emit ItemReceived();
56         state = State.Inactive;
57         buyer.transfer(val);
58         seller.transfer(address(this).balance);
59     }
```

Compiler version

Name

State variables

Constructor

Modifiers

Events

Functions

Dealing with Smart Contracts - Issues

Learning Curve

- Alharby, M., Aldweesh, A., & van Moorsel, A. (2018). Blockchain-based smart contracts: A systematic mapping study of academic research (2018). In *Proceedings of the 2018 International Conference on Cloud Computing, Big Data and Blockchain*.

Security Issues

- Mavridou, A., & Laszka, A. (2018, February). Designing secure ethereum smart contracts: A finite state machine based approach. In *International Conference on Financial Cryptography and Data Security* (pp. 523-540). Springer, Berlin, Heidelberg.

IT – Business Gap

- Mik, E. (2017). Smart contracts: terminology, technical limitations and real world complexity. *Law, Innovation and Technology, 9*(2), 269-300.
- Bosu, A., Iqbal, A., Shahriyar, R., & Chakraborty, P. (2019). Understanding the motivations, challenges and needs of blockchain software developers: A survey. *Empirical Software Engineering, 24*(4), 2636-2673.

"In other words, they're code that does what it's been programmed to do.

If the **business rules** ... have been defined badly and/or the programmer doesn't do a good job, the result is going to be a mess, and, even if programmed correctly, a smart contract isn't smart – it just functions as **designed**."

What's a smart contract (and how does it work)?
Computer World, Jul 29 (2019)

Dealing with Smart Contracts – Proposals (I)

- [Formal] verification of Smart Contracts

Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, A., Gonthier, G., Kobeissi, N., ... & Zanella-Béguelin, S. (2016, October). Formal verification of smart contracts: Short paper. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security* (pp. 91-96). ACM.

Bragagnolo, S., Rocha, H., Denker, M., & Ducasse, S. (2018, March). SmartInspect: solidity smart contract inspector. In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)* (pp. 9-18). IEEE.

- DSL-based
  - Legal principles-based DSL (Adico-Solidity).
  - Natural language-based (SmaCoNat)

Frantz, C. K., & Nowostawski, M. (2016, September). From institutions to code: Towards automated generation of smart contracts. In *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS* W)* (pp. 210-215). IEEE.

Regnath, E., & Steinhorst, S. (2018, September). SmaCoNat: Smart Contracts in Natural Language. In *2018 Forum on Specification & Design Languages (FDL)* (pp. 5-16). IEEE.

- Templates for Smart Contracts

Clack, C. D., Bakshi, V. A., & Braine, L. (2016). Smart contract templates: foundations, design landscape and research directions. *arXiv preprint arXiv:1608.00771*.

Dealing with Smart Contracts – Proposals (II)

- MDE-based
  - Both use MDE to map the business process (BPMN) into a smart contract.
  - Lorikeet need to extend the BPMN notation (2 elements)

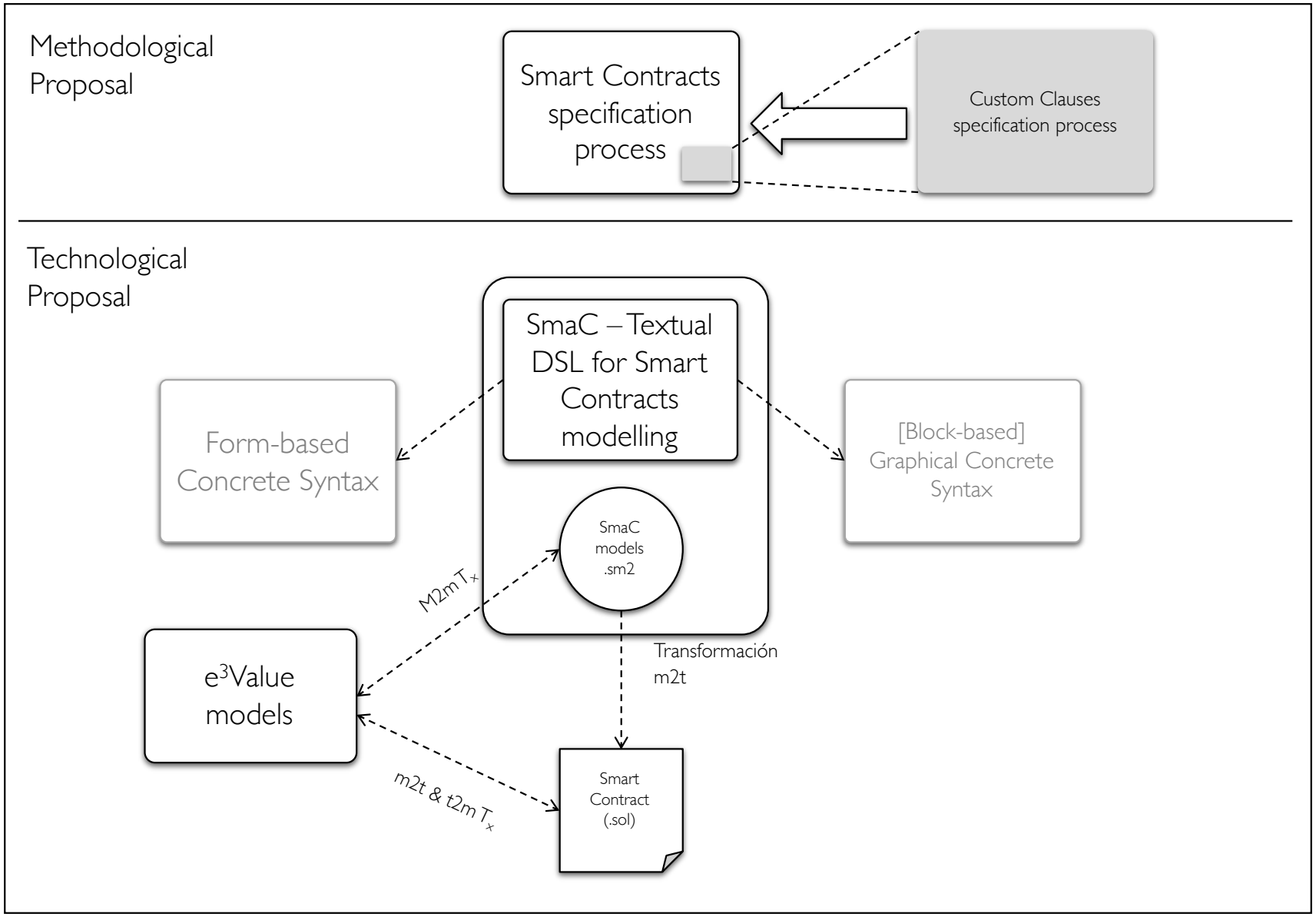
López-Pintado, O., García-Bañuelos, L., Dumas, M., & Weber, I. (2017, September). Caterpillar: A Blockchain-Based Business Process Management System. In *BPM (Demos)*.

Tran, A. B., Lu, Q., & Weber, I. (2018). Lorikeet: A Model-Driven Engineering Tool for Blockchain-Based Business Process Execution and Asset Management. In *BPM (Dissertation/Demos/Industry)* (pp. 56-60).

Agenda

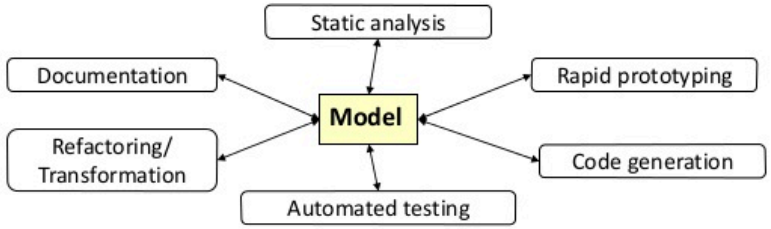
1. Blockchain
2. Smart Contracts
3. Bringing models to the rescue
4. The road ahead

Our proposal



MDE Advantages

- Define and enrich customized textual structures
  - E.g: gas control in loops to avoid
- Reduce the learning curve
  - Auto-completion
  - Syntactical validation
  - QuickFixes
  - Good practices
  - Auto-documentation …
- Development of technological bridges
  - Close the gap between business professionals and developers

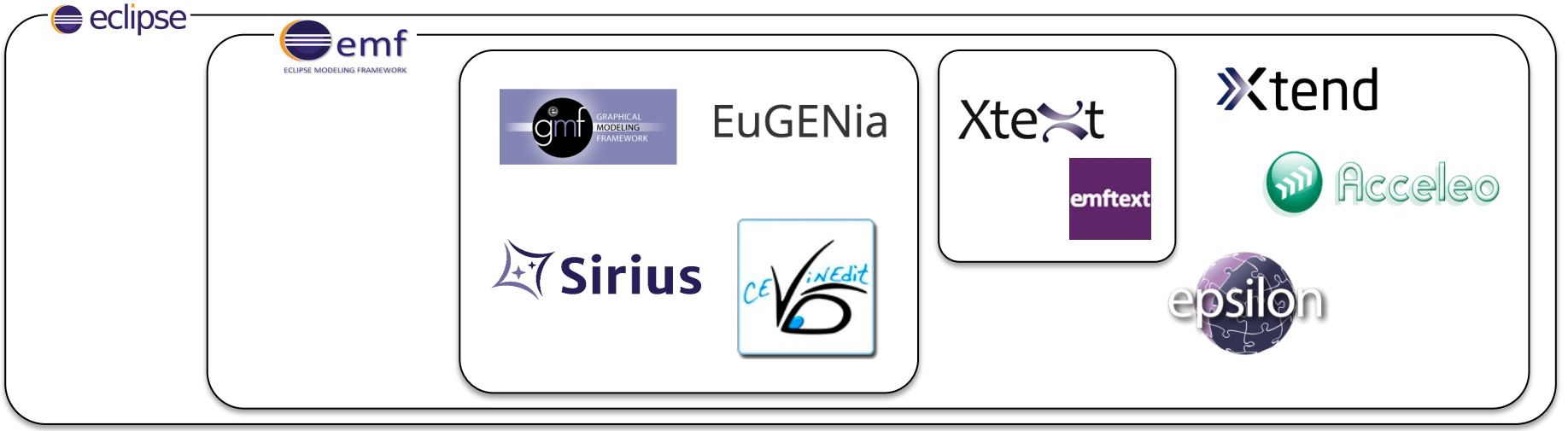


[Illustration by Bernhard Rumpe]

Cabot, J. Lightweight Model-Driven Engineeering. Les journées nationales du GDR GPL. Jun 15, 2017

Xtext

- What is Xtext?
  - Framework for textual DSLs development
  - Xtend (Java-like) for the development of validations, quickfixes, etc.
  - Ecore metamodel automatically generated from the grammar.



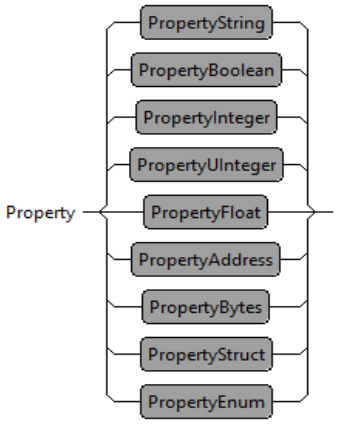
Develop a textual language using Xtext(I)

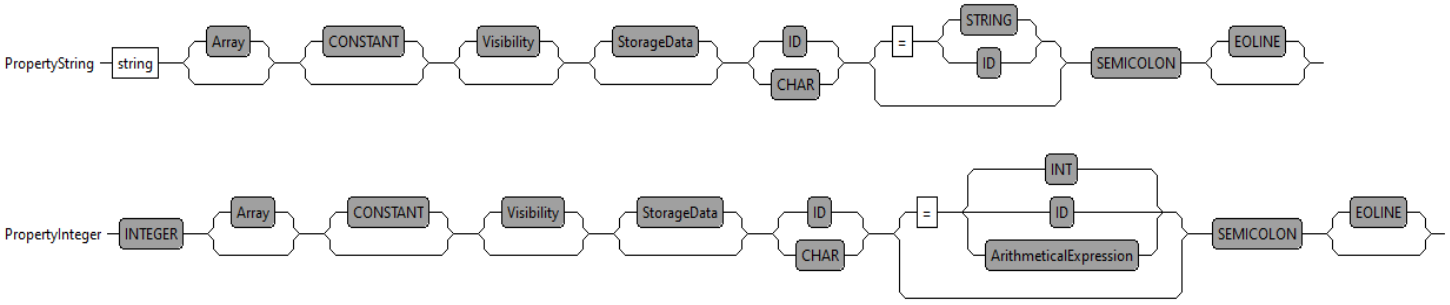
- How to develop a textual language?
  1. Write the grammar
     a) Define the terminals.

```
terminal SEMICOLON returns ecore::EChar:
    ';'
;

terminal DOT returns ecore::EChar:
    '.'
;
```

     b) Define the rules.

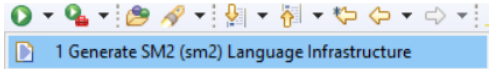
```
Property returns ecore::EAttribute:
    PropertyString|PropertyBoolean|PropertyInteger|PropertyUInteger|PropertyFloat|PropertyAddress|PropertyBytes|PropertyStruct|PropertyEnum
;

PropertyString:
    type= "string" Array? CONSTANT? visibility = Visibility? (storageData = StorageData)? (namePropertyString = ID|CHAR) ('=' inicialization = (STRING|ID))? SEMICOLON EOLINE?
;

PropertyInteger:
    type = INTEGER  Array? CONSTANT? visibility = Visibility? (storageData = StorageData)? (namePropertyInteger = ID|CHAR) ('='  (INT|ID|ArithmeticalExpression))? SEMICOLON EOLINE?
;
```

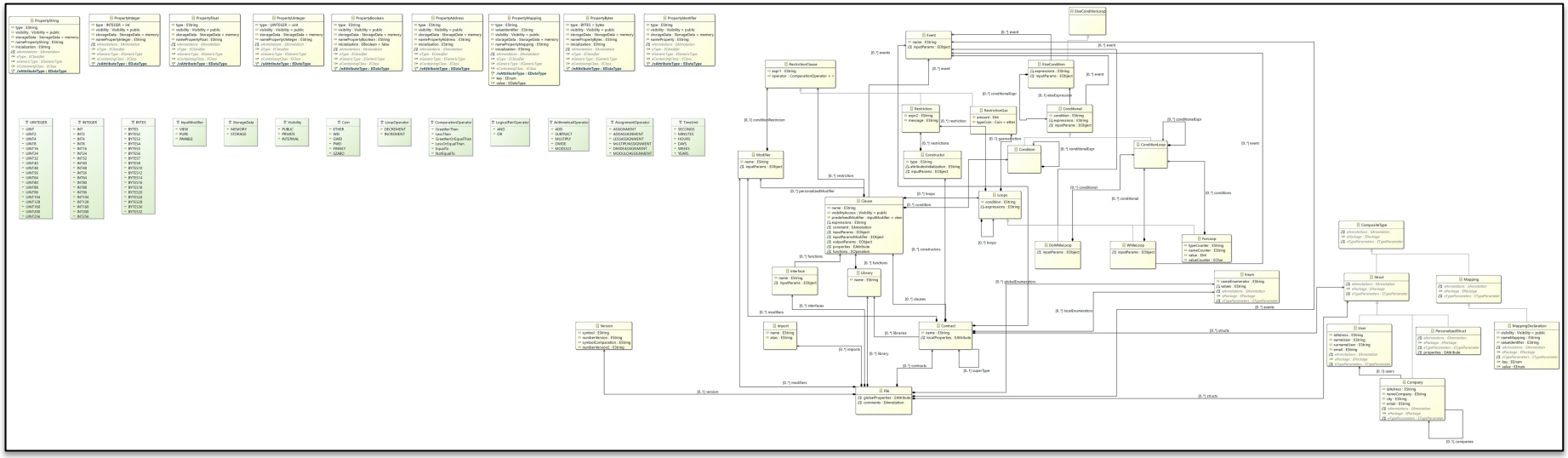




Develop a textual language using Xtext(II)

- How to develop a textual language?
  3. Generate language artifacts.





Develop a textual language using Xtext(II)

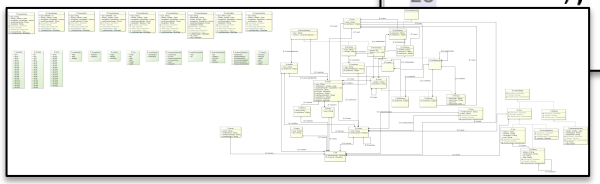
- How to develop a textual language?
  4. Run the Generated Eclipse plug-in.

*My.sce

```
1  pragma solidity >=0.4.22 <0.6.0;
2
3  contract Purchase {
4      uint public val;
5      address payable public seller;
6      address payable public buyer;
7      enum State { Created, Locked, Inactive }
8      State public state;
9
10     // Ensure that `msg.value` is an even number.
11     // Division will truncate if it is an odd number.
12     // Check via multiplication that it wasn't an odd number.
13     constructor() public payable {
14         seller = msg.sender;
15         val = msg.value / 2;
16         require((2 * val) == msg.value, "Value has to be even.");
17     }
18
19     modifier condition(bool _condition) {
20         require(_condition);
21         _;
22     }
23
24     modifier onlyBuyer() {
25         require(
26             msg.sender == buyer,
27             "Only buyer can call this."
28         );
```

Outline

- File
  - >=
  - Purchase
    - public
      - 2 * val
    - condition
      - _condition
    - onlyBuyer
      - msg.sender
    - onlySeller
      - msg.sender
    - inState
      - state
    - Aborted
    - PurchaseConfirmed
    - ItemReceived
    - abort
    - confirmPurchase
    - confirmReceived



Conforms To

Develop a textual language using Xtext(II)

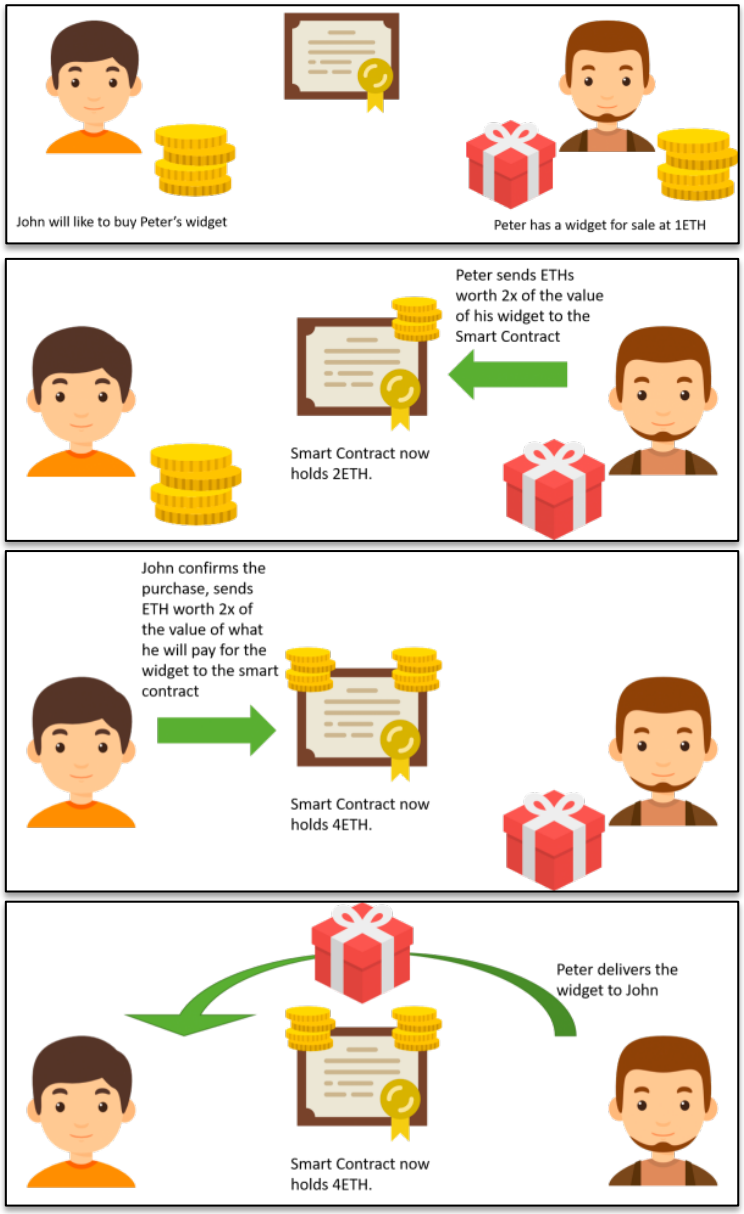
- How to develop a textual language?
  5. [Generate Code Generator - Xtend]
  6. [Unit Testing]
  7. [Creating Custom Validation Rules]

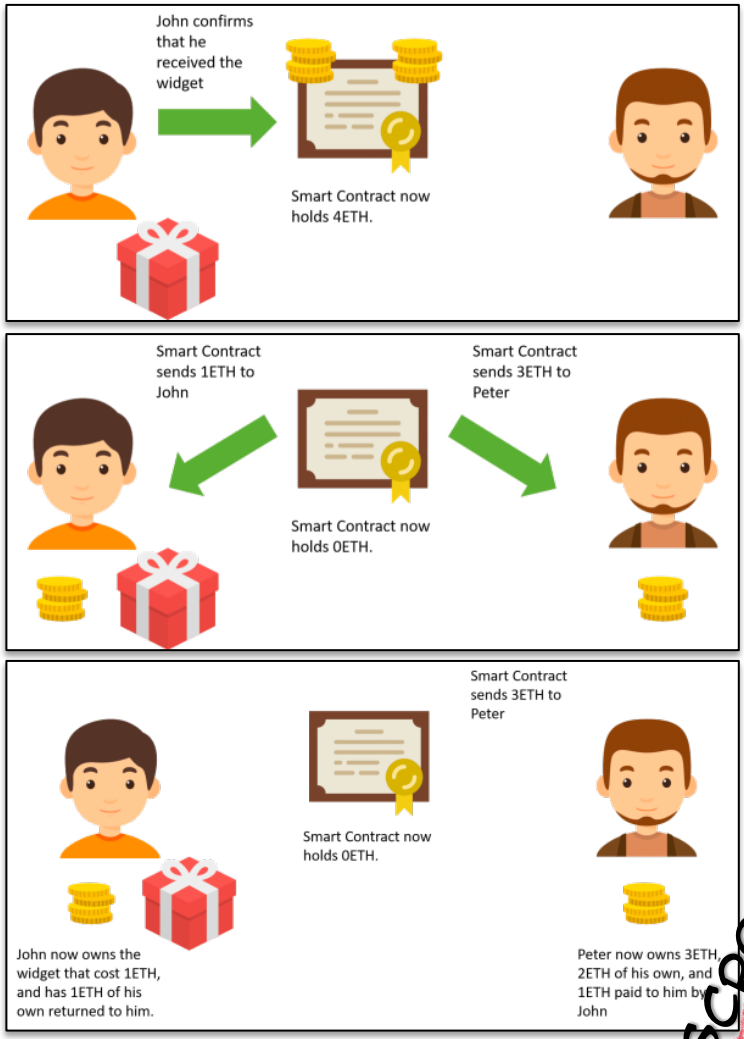
```
@Check
def checkContractStartsWithCapital(Contract imports) {
    if (!Character.isUpperCase(imports.name.charAt(0))) {
        error('Contract´s name should start with a capital',
                SM2Package.Literals.CONTRACT__NAME,
                INVALID_NAME)
    }
}
```

Including quickfixes

```
@Fix(SM2Validator.INVALID_NAME)
def capitalizeName(Issue issue, IssueResolutionAcceptor acceptor) {
    acceptor.accept(issue, 'Capitalize name', 'Capitalize the name.', 'upcase.png') [
        context |
        val xtextDocument = context.xtextDocument
        val firstLetter = xtextDocument.get(issue.offset, 1)
        xtextDocument.replace(issue.offset, 1, firstLetter.toUpperCase)
    ]
}
```

Example: Safe Remote Purchase





https://jacksonng.org/Safe-Remote-Purchase-1

ESCROW SERVICES

- SmaC in action
- Safe Remote Purchase in Solidity documentation

Business (Process) Modeling

Canvas

e³value

Service Blueprint

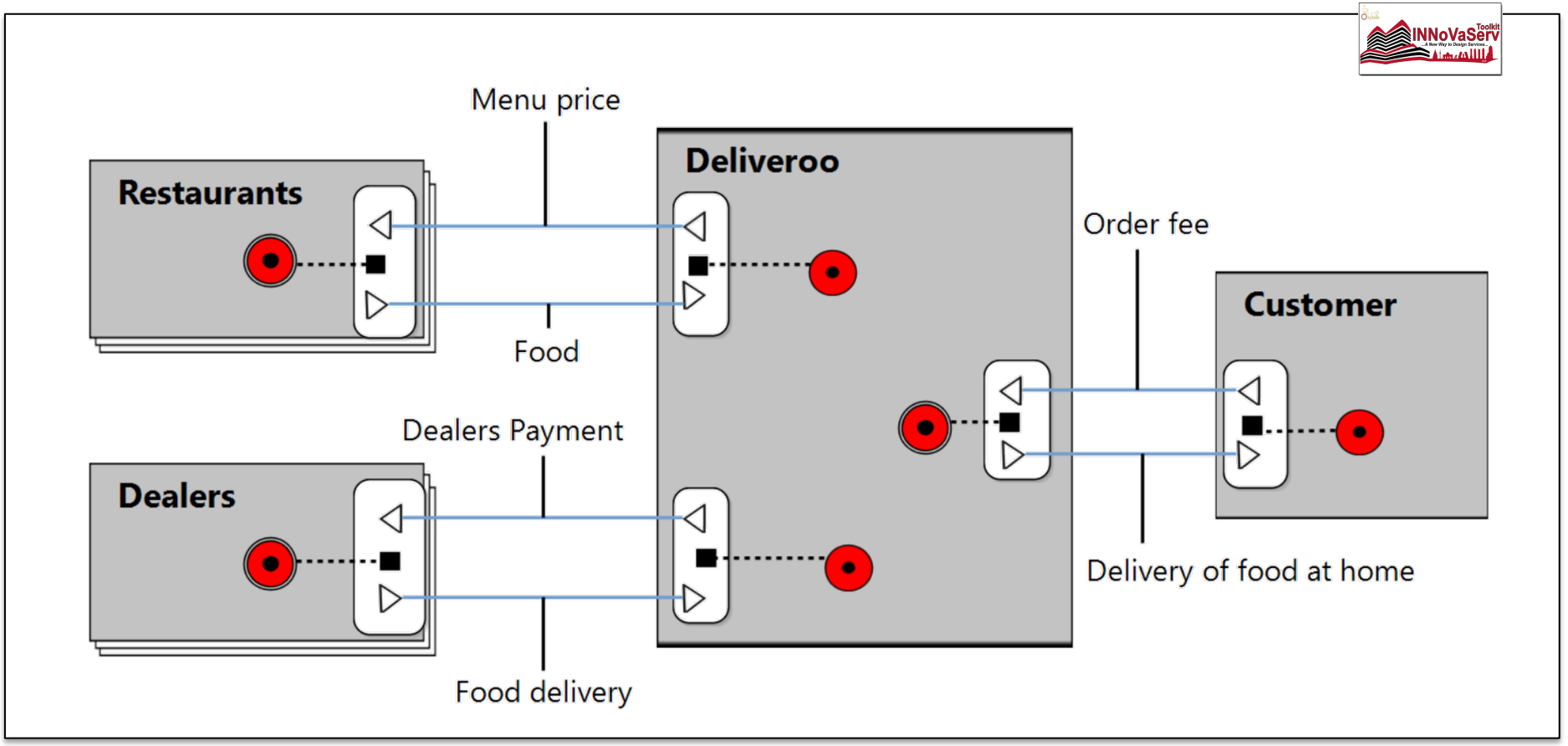
PCN

BPMN

e³value

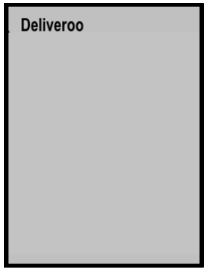
- Business modeling notation
  - Focused on representing the value interchanges between the different actors involved in the provision of a service.

Gordijn, J., & Akkermans, H. (2001). Designing and evaluating e-business models. *IEEE intelligent Systems*, (4), 11-17.

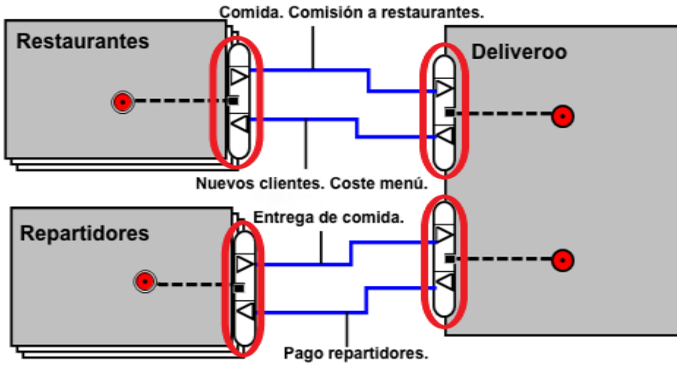


e3value in a nutshell

- Actors
  - An entity that carries out value activities that allow him/her/it to increase … profit or utility

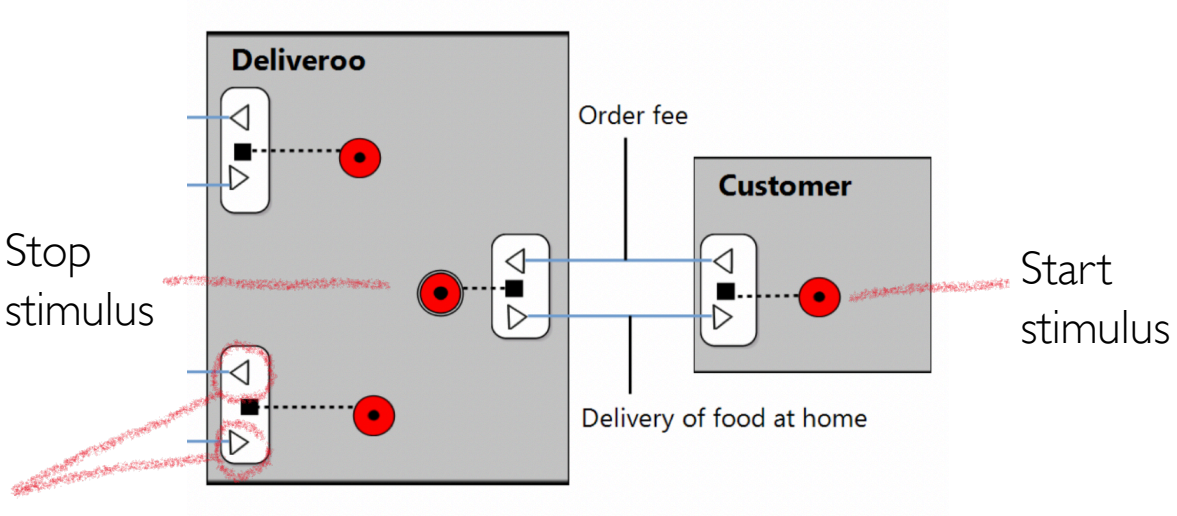


- Value Interface
  - Group the ports through which the actor is willing to make value interchanges
  - A form of representation of economic reciprocity of value between actors.



e3value in a nutshell

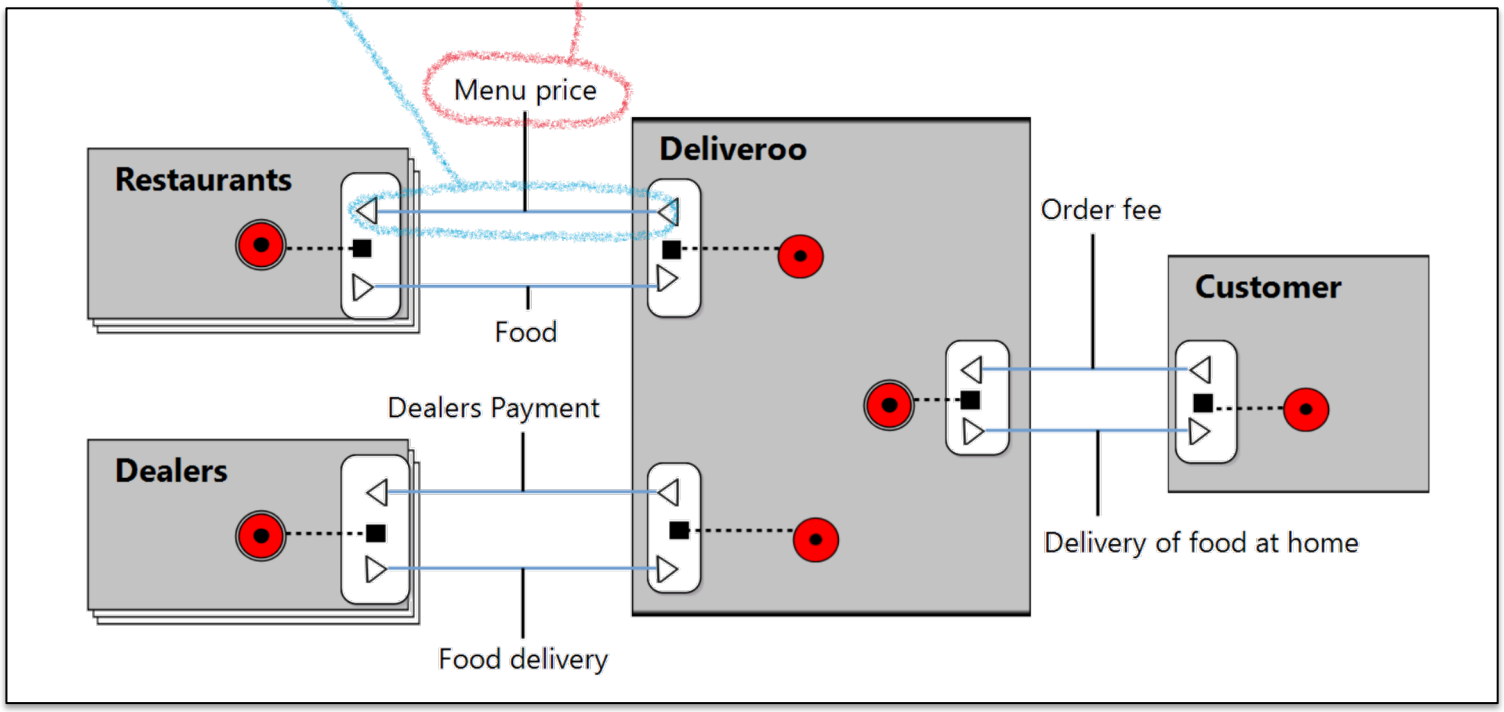
- Stimulus
  - Events caused by an actor, trigger come value exchange.
    - E.g.: Deliveroo's customer is hungry
  - Two types: Start stimulus y Stop stimulus.



- Value Ports
  - Used by an actor to request **value objects** to or from its environment (directional)

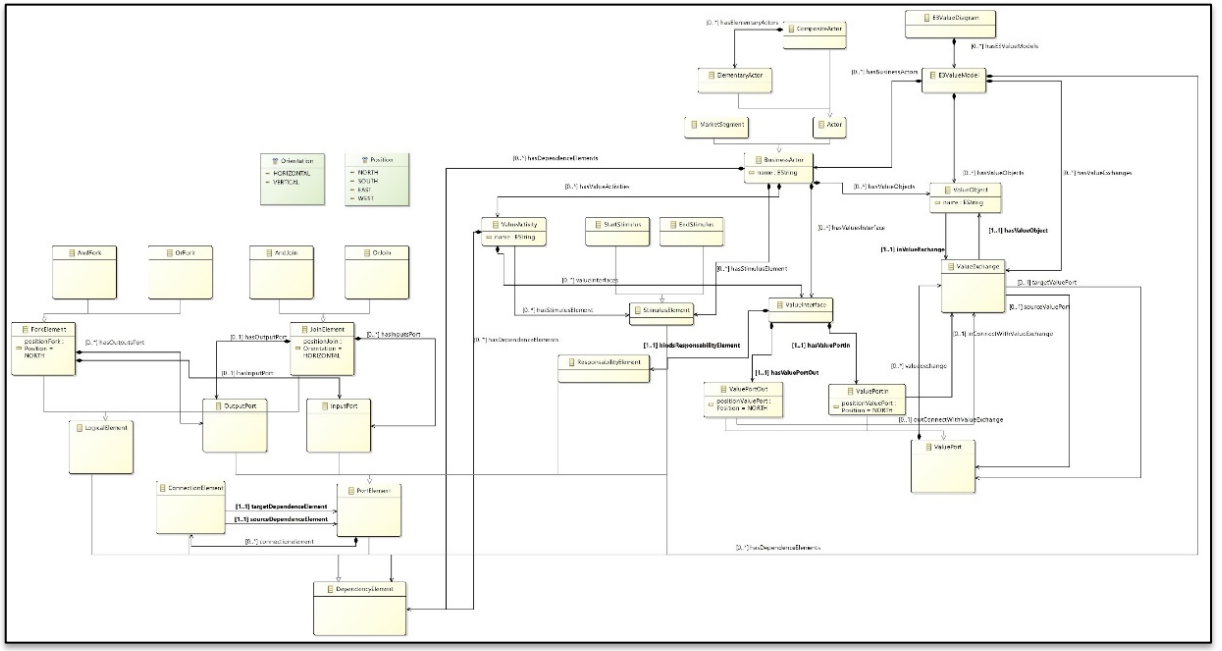
e3value in a nutshell

- Value Transfer
  - Connect two value ports with each other in order to enable value objects exchange
- Value Object
  - Satisfies a particular need or is used to produce other value objects.

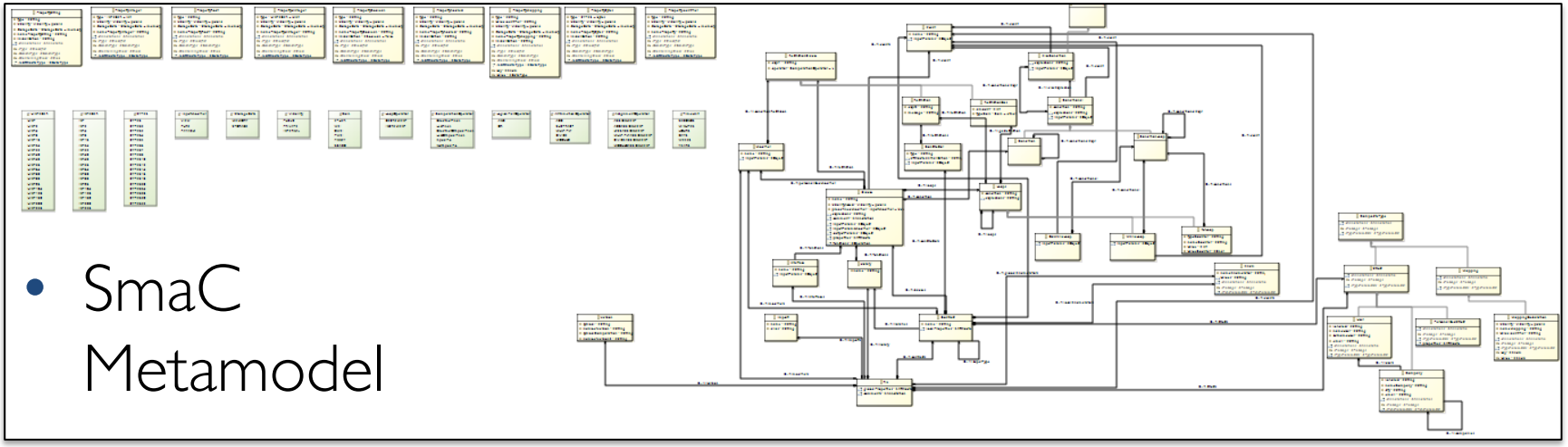


Correspondences

- e³value Metamodel

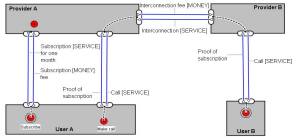


- SmaC Metamodel



Correspondences Analysis

Address / User (Personalized Struct)	⟷	Actors
Address Array / Company (Personalized Struct)	⟷	Market segment
Cryptocoins, Services, Assets, Advices	⟷	Value Object
Events	⟷	Stimulus
Smart contract	<---->	Value Interface
Functions	⟷	Value Port
		Value Transfer



⟷ Compatible

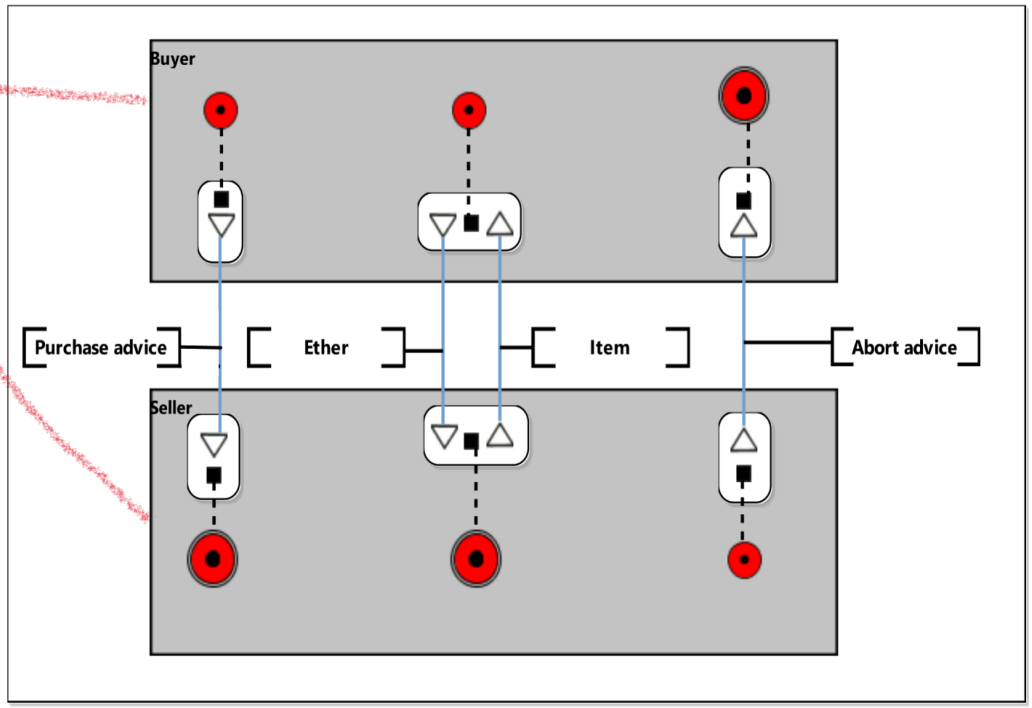
<----> Partially compatible

Correspondences Analysis

- Adress ⇔ Actors

```
1  pragma solidity >=0.4.22 <0.6.0;
2
3  contract Purchase{
4
5      uint public val;
6      address payable public seller;
7      address payable public buyer;
8
9      enum State{Created,Locked,Inactive};
10     State public state;
11
12     constructor()public{
13         seller = msg.sender;
14         val = msg.value /2;
15         require((2 * val) == msg.value,"Value has to be even.");
16     }
17
18     modifier condition(bool condition){
19         require(condition);
20         _;
21     }
22
23     modifier onlyBuyer(){
24         require(msg.sender == buyer,"Only buyer can call this.");
25         _;
26     }
27
28     modifier onlySeller(){
29         require(msg.sender == seller,"Only seller can call this.");
30         _;
31     }
32
33     modifier inState(State _state){
34         require(state == _state);
35         _;
36     }
37
38     event Aborted();
39     event PurchaseConfirmed();
40     event ItemReceived();
41
42     function abort()public onlySeller inState(State.Created){
43         emit Aborted();
44         state = State.Inactive;
45         seller.transfer(address(this).balance);
46     }
47
48     function confirmPurchase() public inState(State.Created) condition(msg.value == (2 * val)) {
49         emit PurchaseConfirmed();
50         buyer = msg.sender;
51         state = State.Locked;
52     }
53
54     function confirmReceived() public onlyBuyer inState(State.Locked){
55         emit ItemReceived();
56         state = State.Inactive;
57         buyer.transfer(val);
58         seller.transfer(address(this).balance);
59     }
```

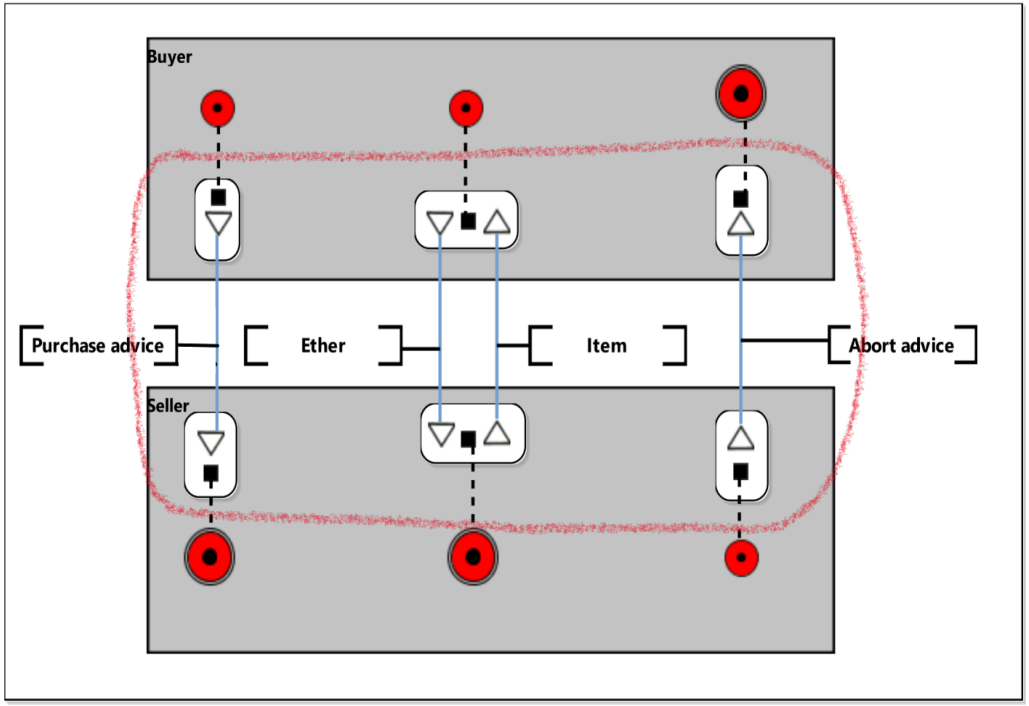
Actors



Correspondences Analysis

- Smart contract ⇄ Value Interface(s)

```
pragma solidity >=0.4.22 <0.6.0;

contract Purchase{

    uint public val;
    address payable public seller;
    address payable public buyer;

    enum State{Created,Locked,Inactive};
    State public state;

    constructor()public{
        seller = msg.sender;
        val = msg.value /2;
        require((2 * val) == msg.value,"Value has to be even.");
    }

    modifier condition(bool condition){
        require(condition);
        _;
    }

    modifier onlyBuyer(){
        require(msg.sender == buyer,"Only buyer can call this.");
        _;
    }

    modifier onlySeller(){
        require(msg.sender == seller,"Only seller can call this.");
        _;
    }

    modifier inState(State _state){
        require(state == _state);
        _;
    }

    event Aborted();
    event PurchaseConfirmed();
    event ItemReceived();

    function abort()public onlySeller inState(State.Created){
        emit Aborted();
        state = State.Inactive;
        seller.transfer(address(this).balance);
    }

    function confirmPurchase() public inState(State.Created) condition(msg.value == (2 * val)) {
        emit PurchaseConfirmed();
        buyer = msg.sender;
        state = State.Locked;
    }

    function confirmReceived() public onlyBuyer inState(State.Locked){
        emit ItemReceived();
        state = State.Inactive;
        buyer.transfer(val);
        seller.transfer(address(this).balance);
    }
```

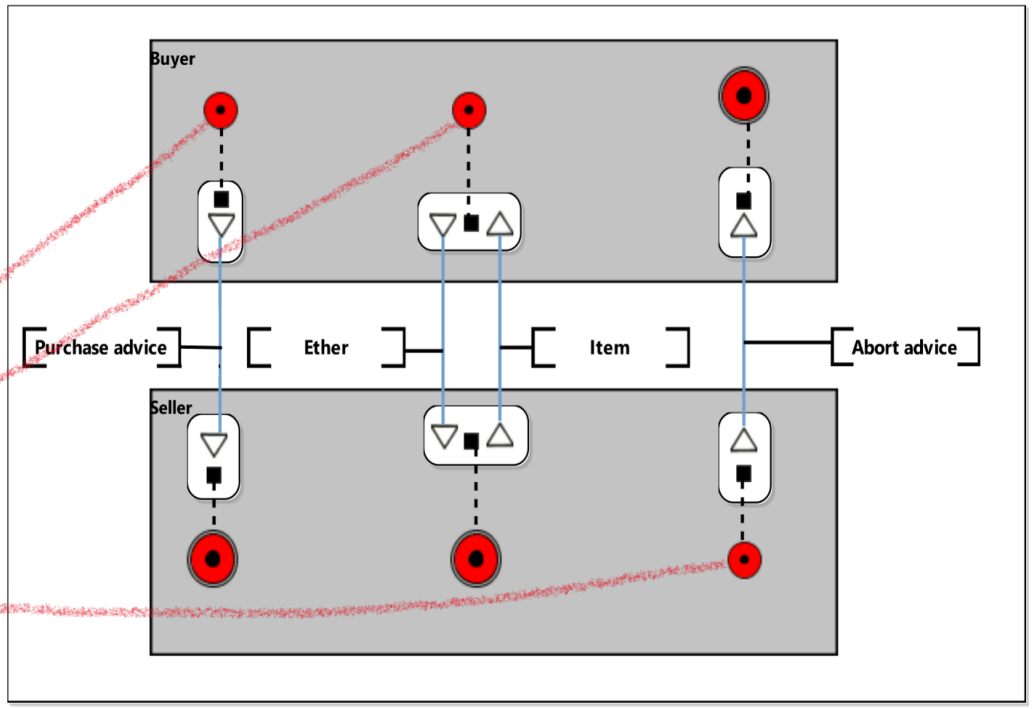


Correspondences Analysis

- Events ⇔ Start stimulus

```
pragma solidity >=0.4.22 <0.6.0;

contract Purchase{

    uint public val;
    address payable public seller;
    address payable public buyer;

    enum State{Created,Locked,Inactive};
    State public state;

    constructor()public{
        seller = msg.sender;
        val = msg.value /2;
        require((2 * val) == msg.value,"Value has to be even.");
    }

    modifier condition(bool condition){
        require(condition);
        _;
    }

    modifier onlyBuyer(){
        require(msg.sender == buyer,"Only buyer can call this.");
        _;
    }

    modifier onlySeller(){
        require(msg.sender == seller,"Only seller can call this.");
        _;
    }

    modifier inState(State _state){
        require(state == _state);
        _;
    }

    event Aborted();
    event PurchaseConfirmed();
    event ItemReceived();

    function abort()public onlySeller inState(State.Created){
        emit Aborted();
        state = State.Inactive;
        seller.transfer(address(this).balance);
    }

    function confirmPurchase() public inState(State.Created) condition(msg.value == (2 * val)) {
        emit PurchaseConfirmed();
        buyer = msg.sender;
        state = State.Locked;
    }

    function confirmReceived() public onlyBuyer inState(State.Locked){
        emit ItemReceived();
        state = State.Inactive;
        buyer.transfer(val);
        seller.transfer(address(this).balance);
    }
```

Events

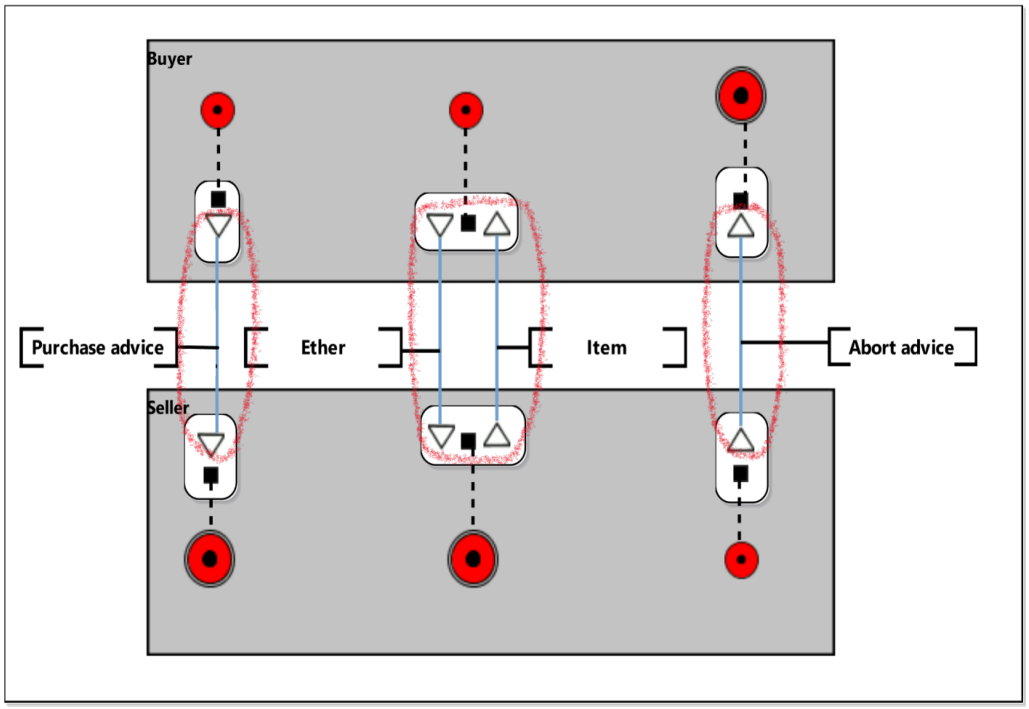


Correspondences Analysis

- Functions ⇔ Value Ports & Value Transfer

```
1  pragma solidity >=0.4.22 <0.6.0;
2
3  contract Purchase{
4
5      uint public val;
6      address payable public seller;
7      address payable public buyer;
8
9      enum State{Created,Locked,Inactive};
10     State public state;
11
12     constructor()public{
13         seller = msg.sender;
14         val = msg.value /2;
15         require((2 * val) == msg.value,"Value has to be even.");
16     }
17
18     modifier condition(bool condition){
19         require(condition);
20         _;
21     }
22
23     modifier onlyBuyer(){
24         require(msg.sender == buyer,"Only buyer can call this.");
25         _;
26     }
27
28     modifier onlySeller(){
29         require(msg.sender == seller,"Only seller can call this.");
30         _;
31     }
32
33     modifier inState(State _state){
34         require(state == _state);
35         _;
36     }
37
38     event Aborted();
39     event PurchaseConfirmed();
40     event ItemReceived();
41
42     function abort()public onlySeller inState(State.Created){
43         emit Aborted();
44         state = State.Inactive;
45         seller.transfer(address(this).balance);
46     }
47
48     function confirmPurchase() public inState(State.Created) condition(msg.value == (2 * val)) {
49         emit PurchaseConfirmed();
50         buyer = msg.sender;
51         state = State.Locked;
52     }
53
54     function confirmReceived() public onlyBuyer inState(State.Locked){
55         emit ItemReceived();
56         state = State.Inactive;
57         buyer.transfer(val);
58         seller.transfer(address(this).balance);
59     }
```

Functions



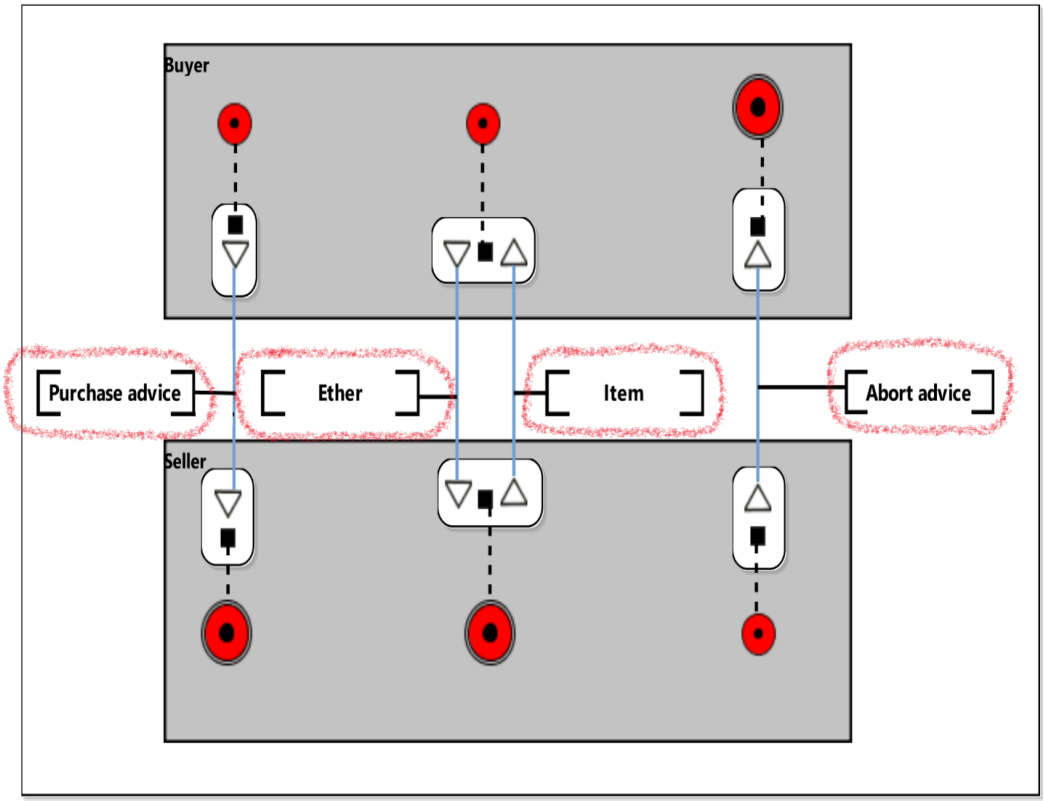
Correspondences Analysis

- Notifications, Badges, Permissions … ⇔ Value Objects

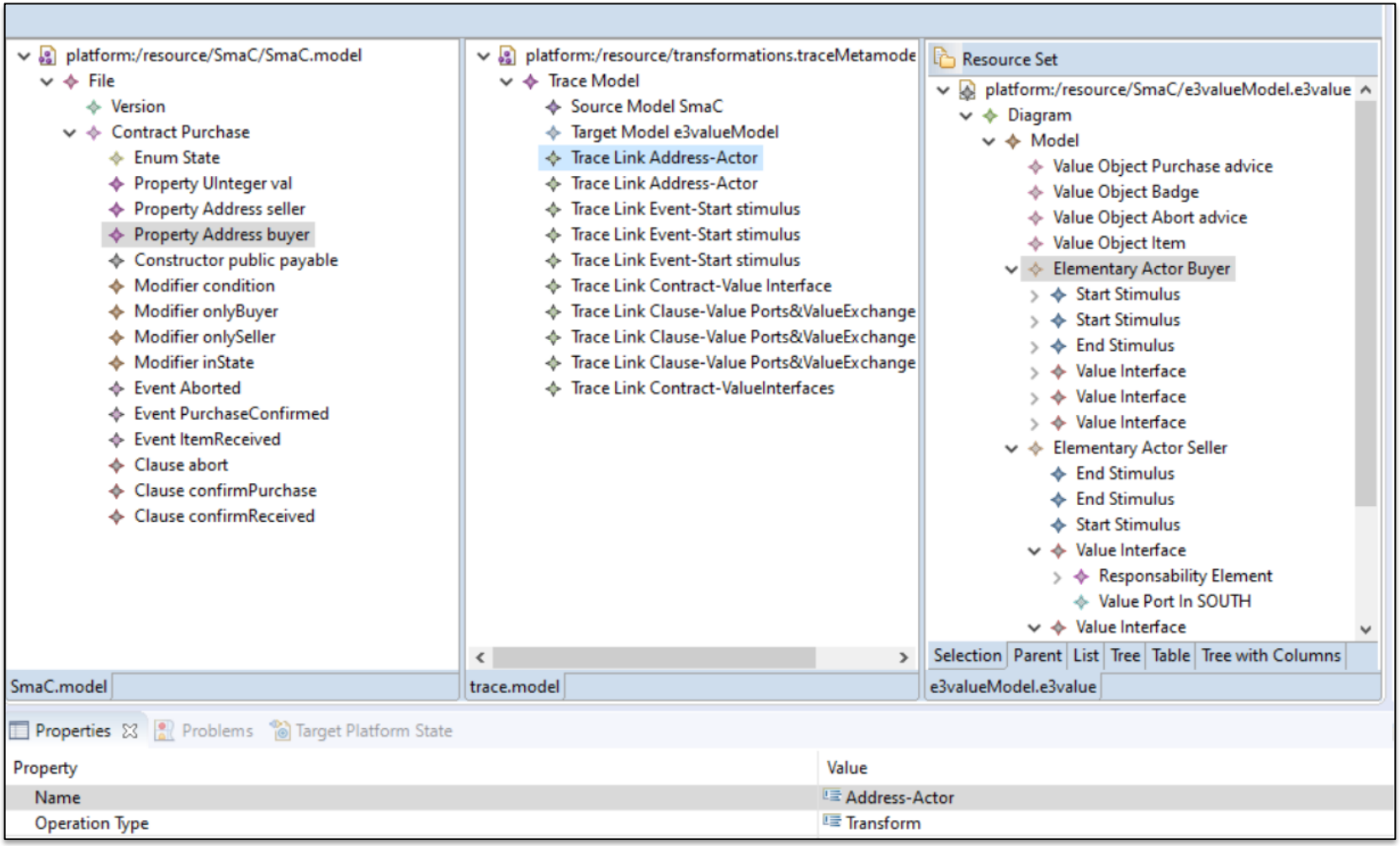
```
pragma solidity >=0.4.22 <0.6.0;

contract Purchase{

    uint public val;
    address payable public seller;
    address payable public buyer;

    enum State{Created,Locked,Inactive};
    State public state;

    constructor()public{
        seller = msg.sender;
        val = msg.value /2;
        require((2 * val) == msg.value,"Value has to be even.");
    }

    modifier condition(bool condition){
        require(condition);
        _;
    }

    modifier onlyBuyer(){
        require(msg.sender == buyer,"Only buyer can call this.");
        _;
    }

    modifier onlySeller(){
        require(msg.sender == seller,"Only seller can call this.");
        _;
    }

    modifier inState(State _state){
        require(state == _state);
        _;
    }

    event Aborted();
    event PurchaseConfirmed();
    event ItemReceived();

    function abort()public onlySeller inState(State.Created){
        emit Aborted();
        state = State.Inactive;
        seller.transfer(address(this).balance);
    }

    function confirmPurchase() public inState(State.Created) condition(msg.value == (2 * val)) {
        emit PurchaseConfirmed();
        buyer = msg.sender;
        state = State.Locked;
    }

    function confirmReceived() public onlyBuyer inState(State.Locked){
        emit ItemReceived();
        state = State.Inactive;
        buyer.transfer(val);
        seller.transfer(address(this).balance);
    }
```

Advice

Badge



Visualization

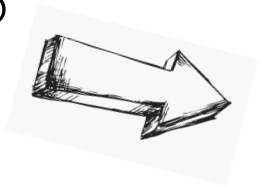


Agenda

1. Blockchain
2. Smart Contracts
3. Bringing models to the rescue
4. The road ahead

Recap

- Blockchain as a way to improve p2p systems
  - Hashing / Asymmetric Cryptography
  - Public ledger as a distributed DDBB
- More recent blockchain networks providing a computational infrastructure
  - Trust-less | Immutability | Transparency
  - Disambiguation + Disintermediation
- Smart Contracts as the way to explode such infrastructure
  - IT – Strategy gap
  - Tooling needed



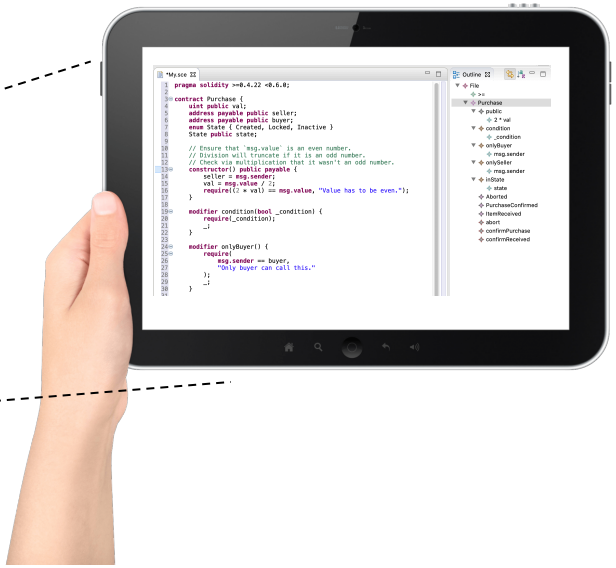
Raise the level of abstraction at which Smart Contracts are developed /designed

Recap

Raise the level of abstraction at which Smart Contracts are developed /designed

MDE TO THE RESCUE

SmaC



The road ahead

- SmaC validation
- Technological Bridges development
  - m2m & m2t transfos.
- Graphical concrete syntaxes development
- Extend e^{3}Value with smart contract elements non directly matched
  - Modifiers.
- Enable automatic deploying mechanisms

Credits

- Delmolino, K., Arnett, M., Kosba, A., Miller, A., & Shi, E. (2016, February). Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In *International Conference on Financial Cryptography and Data Security* (pp. 79-94). Springer, Berlin, Heidelberg.
- Drescher, D. Blockchain Basics: A Non-technical Introduction in 25 Steps, 1st edn. Apress, Frankfurt am Main (2017).
- Escrow Service as a Smart Contract: The Business Logic. Jackson Ng. May 20, 2018. https://jacksonng.org/Safe-Remote-Purchase-1
- Gordijn, J., & Akkermans, H. (2001). Designing and evaluating e-business models. *IEEE intelligent Systems*, (4), 11-17.
- Xtext Documentation: https://www.eclipse.org/Xtext/documentation/
- Xu, X., Weber, I., & Staples, M. (2019). *Architecture for blockchain applications* (pp. 1-307). Berlin, Germany: Springer.