Modelling and Implementation of Access Control Mechanisms in Ethereum Smart Contracts

Thomas Hain, 01/20/2020, Master's Thesis - Final Presentation

Chair of Software Engineering for Business Information Systems (sebis)
Faculty of Informatics
Technische Universität München
wwwmatthes.in.tum.de

Outline

Motivation

High financial incentive to attack Smart Contracts

- Smart Contracts often contain cryptocurrency
- Alteration of currency balances needs to be protected

Low Adaptation of Access Control Mechanisms in Ethereum

Existing solutions are rather limited

⇒ potential to advance the community's understanding of Access Control

Using existing approaches in distributed environment

⇒ Testing Access Control's historical advances against Blockchain technology

Data Privacy is heavily debated

⇒ Discussion even involves Ethereum's founder Vitalik Buterin

Outline

Motivation

Research Questions

Blockchain Basics

Approach

Evaluation

Conclusion & Future Work

Research Questions

1. What are current challenges regarding the implementation of access control on a Blockchain?
2. What is the current state of implementations regarding access control in Solidity?
3. Which advantages does using Blockchain technology provide for access control?
4. How can an extendable access control system be modelled and implemented?

Outline

Motivation

Research Questions

Blockchain Basics

Approach

Evaluation

Conclusion & Future Work

Blockchain Basics

Ethereum and Bitcoin are known as public Blockchains

⇒ All messages are passed between nodes within a peer to peer network

Ethereum is based around **accounts** identified by a **public key**

- Externally Owned Accounts (e.g. a user)
- Contract Accounts

Smart Contracts are stack-based **executable programs** stored on Ethereum's Chain

⇒ Messages between accounts are **not encrypted** and can be read by **every node**

Blockchain Basics

The information they carry includes:

- Transfer of currency
- Function calls to interfaces of Smart Contracts
- Arbitrary data

These messages are called:

- transactions (originating from EOA)
- messages (originating from contract)

Both contract accounts and EOAs can hold and transfer currency

Communication is stored within “Blocks”
A Block is a data structure containing integrity protected transactions

Chaining Blocks these together leads to the name “Blockchain”

Blockchain Basics

Each node holds a copy of the Blockchain
⇒ Multiple coexisting chains on different nodes

The mechanism on how they agree on a state is known as finding **“consensus”**

Thus the network shares understanding

- about how much balance each account carries
- the general state of data within the network*

This includes the **state of contracts** stored on the Blockchain

⇒ Blockchain’s maximize **transparency** (consensus) and **availability** (peer to peer)

But: conflict between transparency and privacy

Outline

Motivation

Research Questions

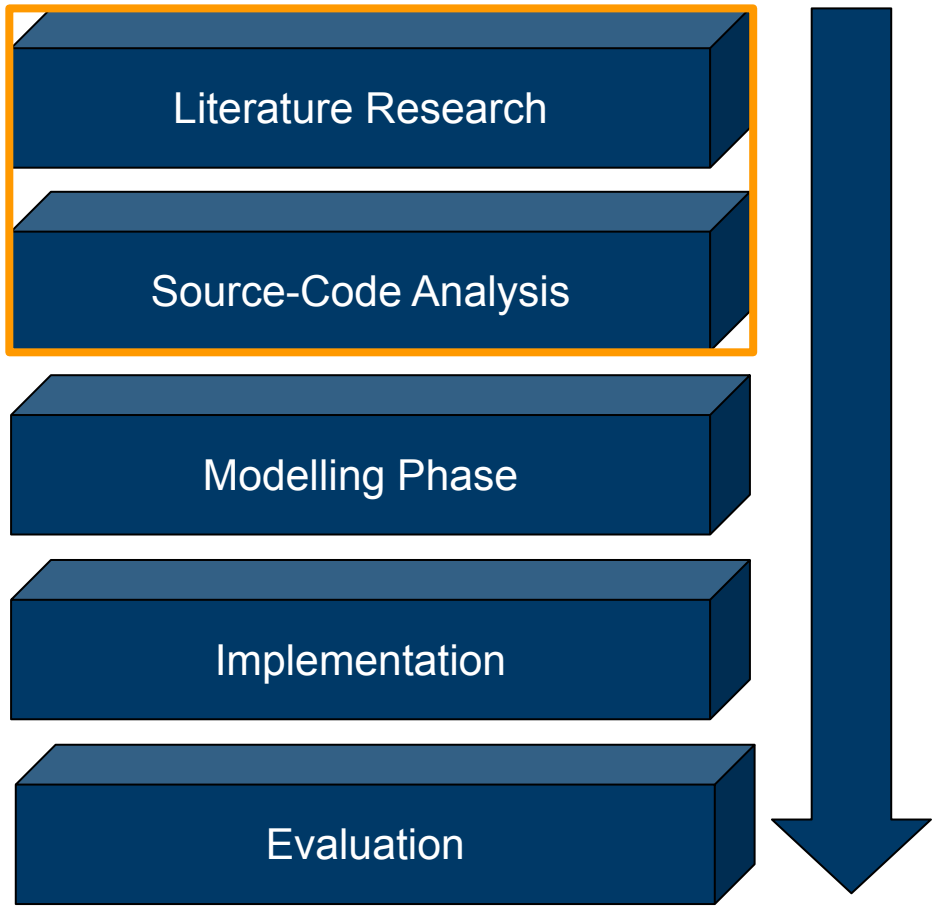
Blockchain Basics

Approach

Evaluation

Conclusion & Future Work

Approach - Literature Research & Source-Code Analysis



Approach - Current State of the Art

Due to their transparency data privacy on Blockchains is heavily debated

Ultimately this lead to

- the introduction of **private and permissioned** Blockchains
  ⇒ possibility to exclude nodes or limit their rights
- heavy use of storing sensitive data off the chain **(“off-chaining”)**

In addition off-chaining is further motivated by **storage and transaction fees**

In Ethereum they are referred to as **“gas costs”**

- They are paid in order to pay for **computational steps**
- Increase with the **amount of data** to be stored on the chain

Transactions **can** include arbitrary data

But: Consensus requires data replication

⇒ Storage is costly

Approach - Current State of the Art

MedRec

- Restricts access to Electronical Health Records by storing SQL queries in contracts
- Queries are linked to Ethereum accounts
- They reflect the permissions of said user for off-chain databases

Attribute-Based Signatures

- Data is encrypted and requires keys linked to certain attributes for decryption
- E.g. (hasActiveSubscription) AND (isStudent))

OpenZeppelin

- Is a library used for different purposes including Access Control
- Provides Smart Contracts for basic role assignment and rights enforcement
- Includes an Implementation of Ownership Pattern

Approach - Current State of the Art

RBAC-SC

- Stores string-based role of users within a publicly accessible contract
- The Smart Contract only serves as a publicly accessible register
- User needs to prove to an institution that he has the account's private key
- Showcases another approach in solving the issue of data privacy
- Enforcement of request is delegated to the institution itself
  ⇒ Possibly off-chain
  ⇒ Request parameters require no publishing to Blockchain
  ⇒ Can be transmitted securely, **e.g. via HTTPS**

Approach - Current State of the Art

Both OpenZeppelin and RBAC-SC use “function modifiers”

- Are used to annotate functions
- They include an underscore statement “_;”
- The modifier’s underscore statement is being replaced by the functions body

```
1  modifier onlyUsers {
2      require(userId[msg.sender] != 0);
3      _;
4  }
```

⇒ Require statement throws an exception if it evaluates to false
Example: Sender of msg is not in array userId

In other words: The user is not registered

Approach - Current State of the Art

Smart Policies
Compiles access policies into executable Smart Contracts: **“Smart Policies”**
Smart Policies encode callable access control decision functions
Enforcement handled by an off-chain Java Program
This program is responsible for updating these contracts

Downsides:
- **Enforcement of Access Control is handled off-chain only**
- Updating policies requires redeployment and deactivation of old Smart Policies
- Programmers require knowledge about additional programming language

Advantages:
- **Policies offer more flexibility than modifiers**
- Based on **XACML language,** which is being maintained by the OASIS Consortium

Approach - Current State of the Art

XACML - eXtensible Access Control Markup Language

- Policies are grouped into **“Policy Sets”**
- Policies contain **“Rules”**
  ⇒ They state effects (e.g. “Permit”)
- Rules contain **“Targets”**
  ⇒ Are matched against **Subjects, Resource & Action**

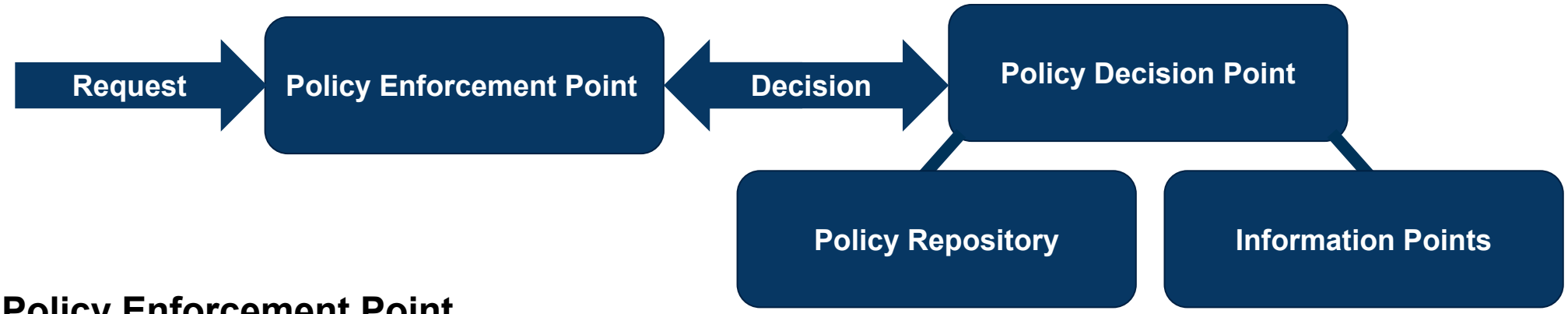
```
1 <Policy>
2     <Rule Effect="Permit">
3         <Target>
4             <Subject "Thomas" />
5             <Resource "Movie: Drive" />
6             <Action "READ" />
7         </Target>
8     </Rule>
9 </Policy>
```

Storing policies on a Blockchain is difficult

- Conflict resolution can be **computationally expensive**
- Rather verbose ⇒ require **much storage**

Approach - Current State of the Art

XACML Architecture



Policy Enforcement Point

- Responsible for handling all incoming requests (**responds**)
- Queries Decision Point
- **Executes request** if Decision Point responds with a grant
  e.g. “delete all movies from database”

Policy Decision Point

- Retrieves Policies from Repository and combines them with Information
- Arrives at a **decision** (based on request, policies and information)
- Informs Enforcement Point of decision (e.g. **grant / deny**)

Approach - Current State of the Art

Results

- There are few adaptable access control solutions
- Many works implement their own strategies
- Data Privacy is enforced by encryption or by communicating off-chain

But one question remains:

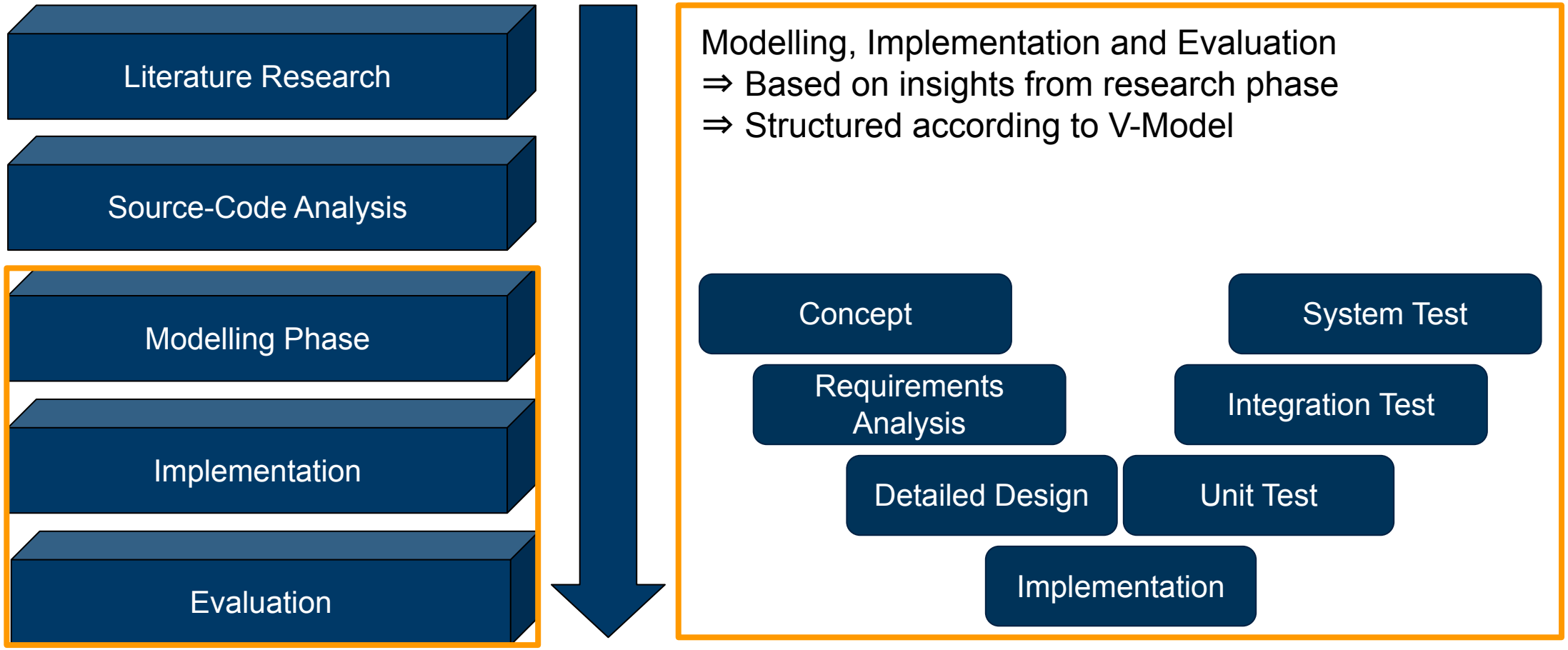
“How can data privacy be combined with Blockchain technology?”

Quorum

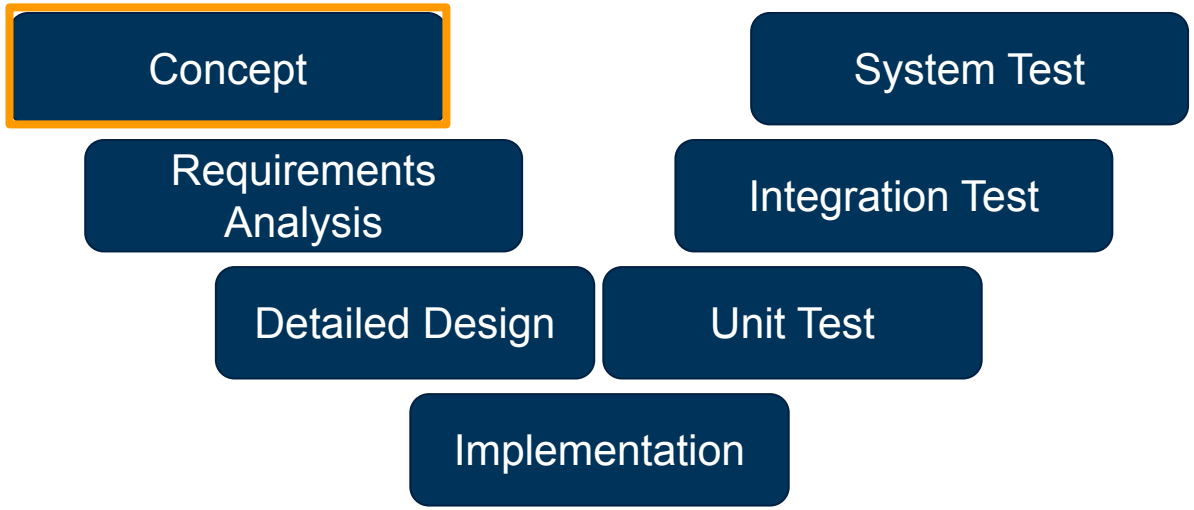
- **“Permissioned Blockchain”**
  ⇒ Allows removal, addition & modification of nodes
  ⇒ Offers hierarchical node organization
- Can be hosted both privately and publicly
- Is based on Ethereum
- Allows private transactions & contracts

However using **private Smart Contracts** leads to loss of transparency
⇒ **breaks public verifiability**

Approach - Modelling, Implementation & Evaluation



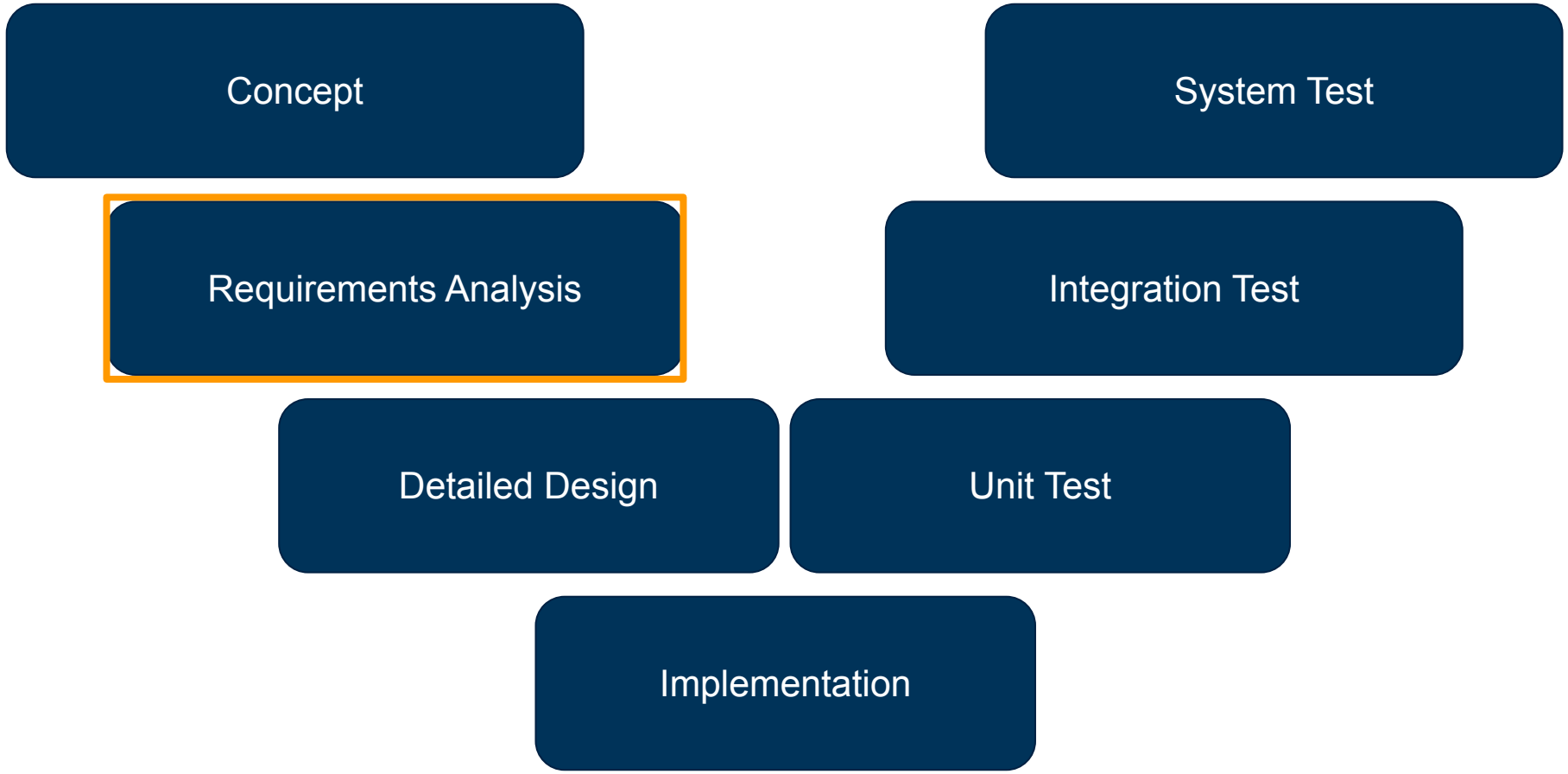
Approach - Modelling, Implementation & Evaluation



Concept

- Flexible Access Control framework
- Allowing both on- and off-chain solutions
- Being fully based on Smart Contracts to support Quorum
- Simplified XACML language - simplified policies

Approach - Modelling, Implementation & Evaluation



Approach - Modelling, Implementation & Evaluation

Functional Requirements

User

UR 1) ...shall be able to send a request
UR 2) ...shall be able to verify the state of his request

Authentication

AU 1) ...shall be able to register a User within a User Storage

Storage Contracts

XS 1) ...shall provide interfaces for CRUD via URIs
XS 2) ...shall notify Subscribers when CRUD data

Non-Functional Requirements

NF 1) Extendability
NF 2) Security
NF 3) Availability

XACML

Enforcement Point

XE 1) ...shall be able to enforce requests on-chain
XE 2) ...shall be able to notify off-chain Enforcement
XE 3) ...shall notify Subscribers about a Grant
XE 4) ...shall notify Subscribers about a Deny

Decision Point

XD 1) ...shall be able to read from a User Storage
XD 2) ...shall be able to include retrieved User information during decision
XD 3) ...shall notify Subscribers about a Deny
XD 4) ...shall notify Subscribers about a Grant
XD 5) ...shall notify Subscribers when its connections to Information Points change
XD 6) ...shall notify Subscribers when its connection to Policy Repository changes

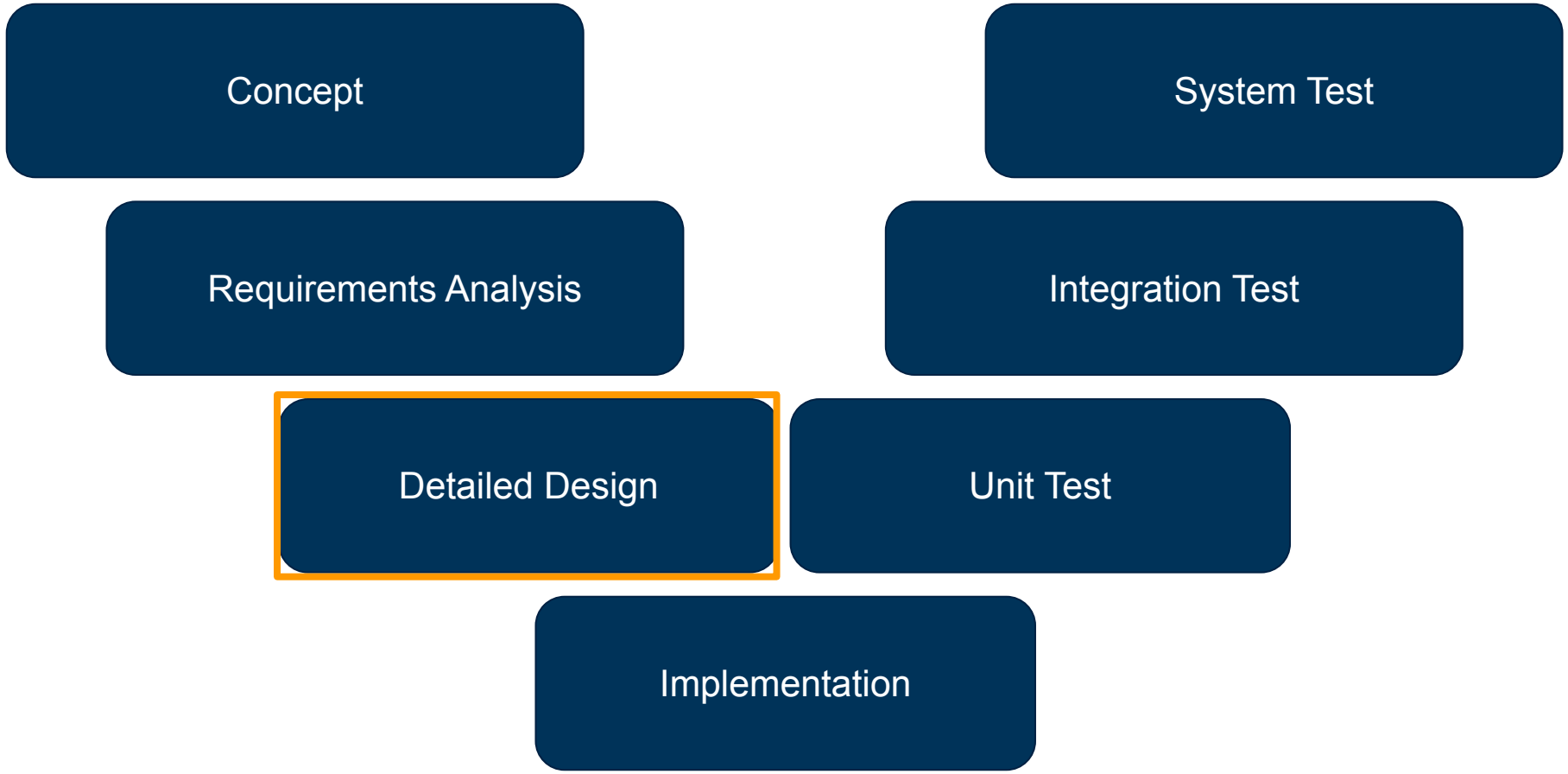
Information Points

XI 1) ...shall be able to respond to the Decision Point

Policy Repository

XP 1) ...shall be able to respond to the Decision Point

Approach - Modelling, Implementation & Evaluation



Approach - Detailed Design

Each of the components' interactions need to be protected,
All functions are exposed to the public
Without protection any user could

- alter the Policy Repository or its address to provide own conditions
- manipulate Information Points to provide false attributes
- ….

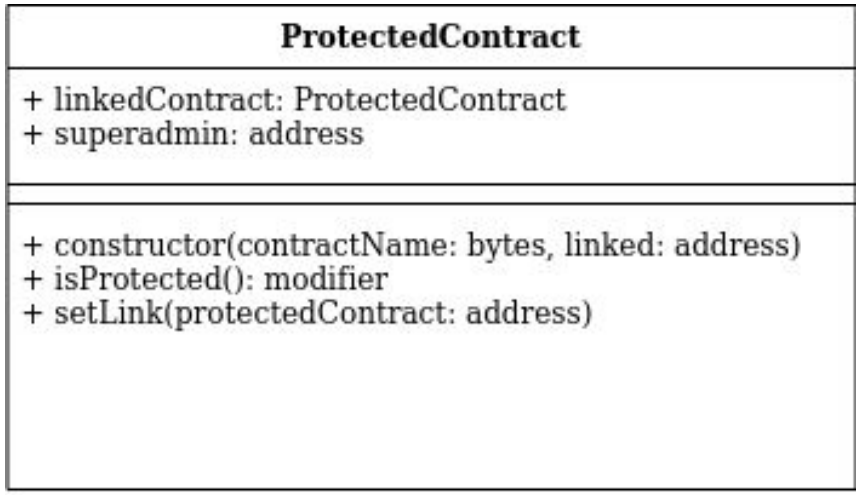
⇒ **State altering** functions need to be exclusively limited to

- Their administrator (the deployer)
- Its interacting components

⇒ **Protected Contract**

Approach - Detailed Design

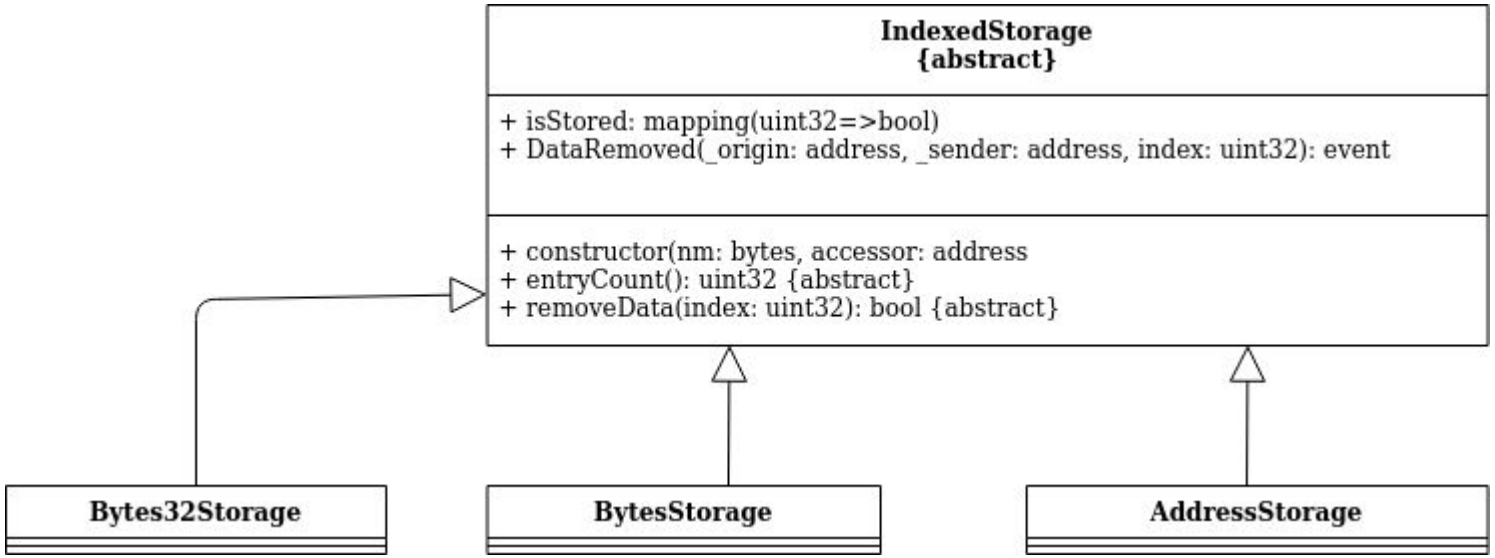
Protected Contract



Consequently each component inherits from “ProtectedContract”
⇒ Implements modifier isProtected for sensitive functionality

Approach - Detailed Design

Indexed Storage



⇒ System requires Storage of Subjects, Resources, Actions and Policies + Conditions
⇒ Their index serve as an identifier
⇒ A user's request is defined as Request(resourceIndex, actionIndex)

Basic storage is necessary if enforcement is **fully transparent** and **on-chain**
However, the system supports off-chain enforcement as well

Approach - Detailed Design

Enforcement Point

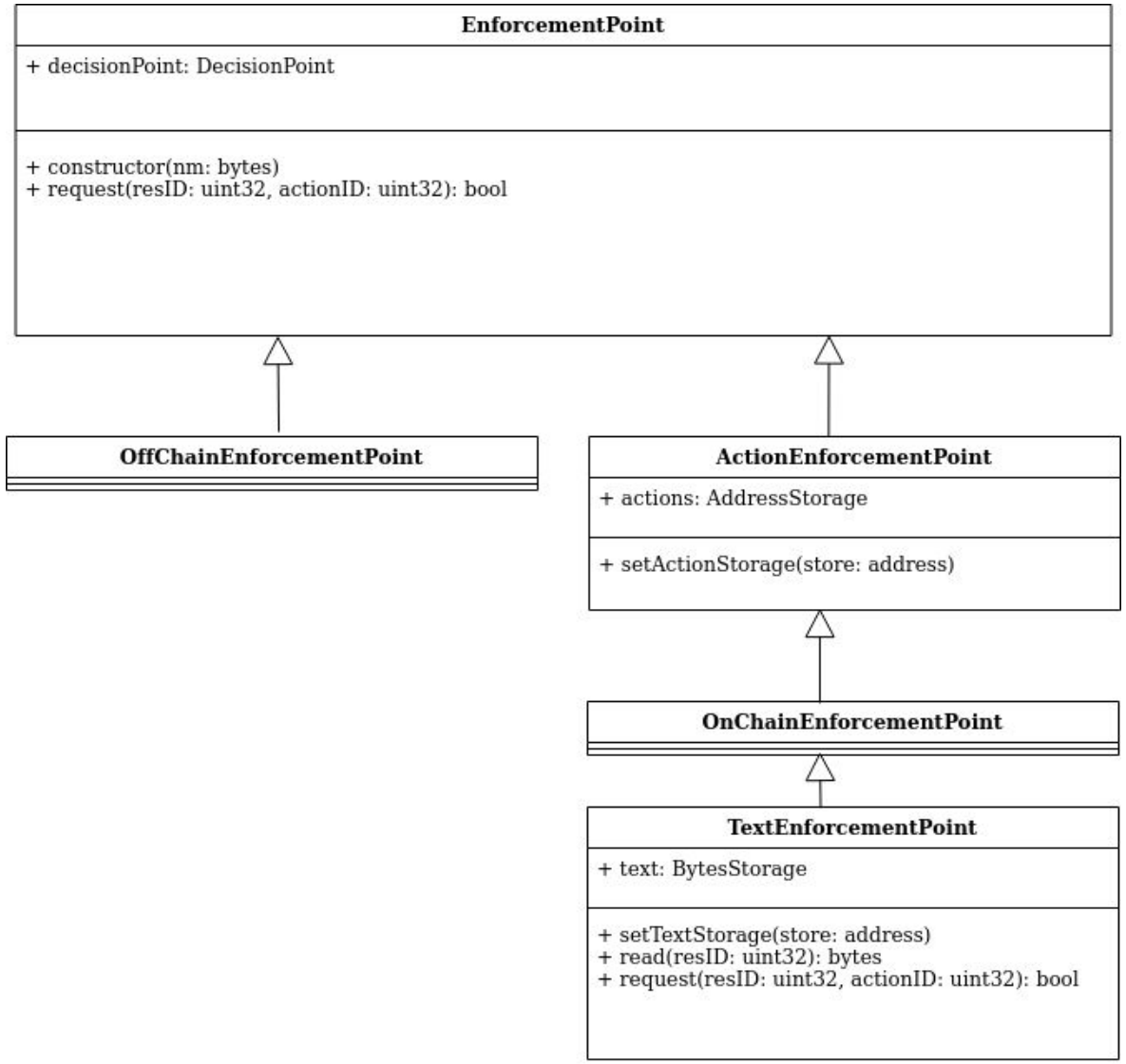
By default the Enforcement Point only responds to a request with the Decision Points Decision

And emits a **grant** or **deny** event

Allows Off-Chain Enforcement programs to subscribe

If full on-chain Enforcement is required
⇒ linking to an **“ActionStorage”** is necessary

An **ActionStorage** is a variation of an AddressStorage
⇒ Stores addresses of contracts
⇒ Contracts implementing an execute-function



Approach - Detailed Design

Decision Point

- The Decision Point was attached a UserStorage (AddressStorage)
- This storage can be used to determine a user’s registration status
- Decision function can be overridden via inheritance
- Possibly queries Policy Repository & Information Points

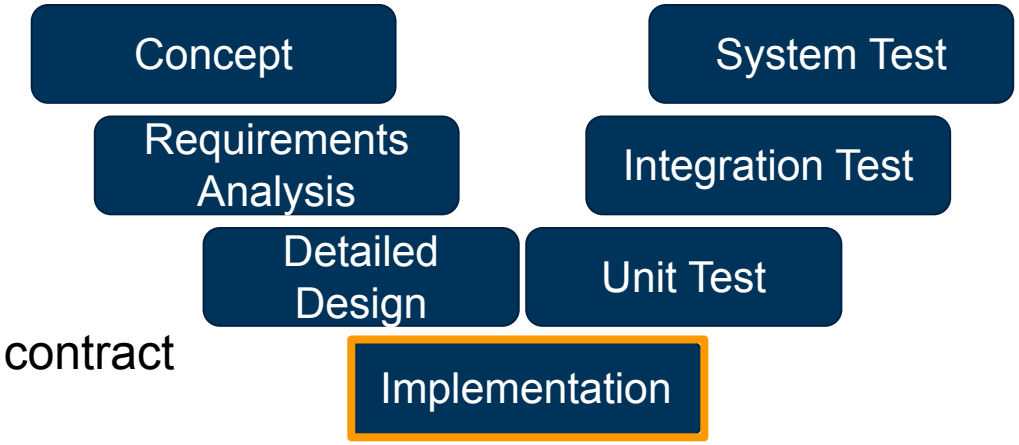
Approach - Implementation

No XACML Policies

Instead bytes32 based attributes + conditions

Can be converted to string (e.g. “isAdmin”)

PolicyRepos & Information Points inherit from same contract
⇒ Implement function processRequest()
⇒ Responds with array of conditions or attributes (bytes32[ ])



Implementation conducted in programming language Solidity **(Version Pragma 0.5.0^)**

In total it includes 26 Smart Contracts

Everything is based on Smart Contracts ⇒ Fully usable in Quorum

Outline

Motivation

Research Questions

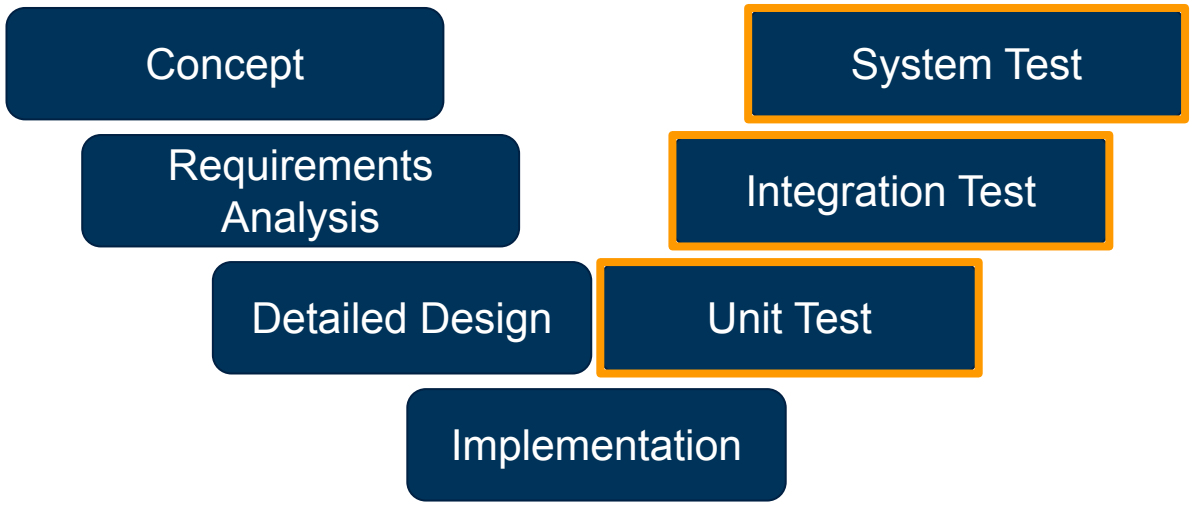
Blockchain Basics

Approach

Evaluation

Conclusion & Future Work

Evaluation: Testing



Testing

- Truffle Testing Suite
- Testing based on Javascript frameworks Mocha & Chai
- Assertion-based

Evaluation: Testing

In total 41 different Test Cases

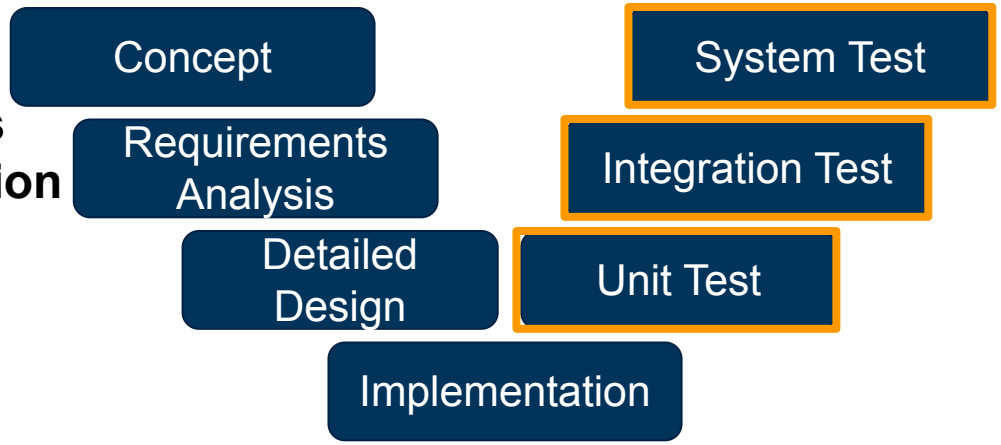
- **Storage Operations**
- **Interactions: Storages & XACML components**
- **Deny if user is unregistered & User Registration**
- **Full System Tests based on inheritance**

Custom Decision Point

⇒ Overridding decision function

Static Information Points & Policy Repository

⇒ Returning fitting / unfitting sets of conditions etc.



Evaluation: Comparison with Smart Policies

	Smart Policies	Implementation
XACML		
X1) Includes Full On-Chain Enforcement	-	+
X2) Supports On-Chain Policy Decisions	+	+
X3) Supports Dynamic Addition of Information Points	~	+
X4) Supports Complex XACML Policies	+	-
Utility		
U1) Includes Basic Authentication Contract	-	+
U2) Supports Resource Abstraction	-	+
U3) Uses Events to notify Subscribers	+	+
U4) Allows public Auditability	+	+

Evaluation: Comparison with Smart Policies

	Smart Policies	Implementation
Privacy		
P1) Can be deployed on Quorum	-	+
P2) Includes Off-Chain Enforcement Point	+	-
Extendability		
E1) Promotes Reusability by Design	-	+
E2) Separation between Private and Public Enforcement	-	+

Outline

Motivation

Research Questions

Blockchain Basics

Approach

Evaluation

Conclusion & Future Work

Conclusion

RQ1) What are current challenges regarding the implementation of access control on a Blockchain?

- Data Privacy vs. public Verifiability (and Auditability)
- Storage and computation Limitations

RQ2) What is the current state of implementations regarding access control in Solidity?

- Few public frameworks and often only modifier based
- Lots of off-chaining
- Sometimes attribute-based encryption

RQ3) Which advantages does using Blockchain technology provide for access control?

- Public verifiability of both execution and enforcement if pure on-chain
- Proof of access right if only decision is on-chain but enforcement off-chain

RQ4) How can an extendable access control system be modelled and implemented?

Inheritance & Interfaces can be leveraged

Future Work

Extension by an Access Control Front End

Allows easier monitoring (events are already in place)

Could potentially resolve conflicting policies

Evolving Encryption Techniques

⇒ Zero-Knowledge Proofs - “zk-SNARKs” are being used for Authentication

Evolution of Solidity

⇒ Structs currently can’t be passed between Contracts ⇒ likely to be included in the future

⇒ Template Programming / Generics are unsupported ⇒ unlikely to be included

Possible Synthesis with MedRec / Smart Policies, ...

⇒ SQL Queries could be managed by on-chain enforcement

⇒ Compiling a “better” policy language with Smart Policies

Literature

1. Samarati, Pierangela, and Sabrina Capitani de Vimercati. "Access control: Policies, models, and mechanisms." *International School on Foundations of Security Analysis and Design*. Springer, Berlin, Heidelberg, 2000.
2. Jason Paul Cruz, Yuichi Kaji, and Naoto Yanai. “RBAC-SC: Role-based access control using smart contract.” In:IEEE Access6 (2018), pp. 12240–12251.issn:21693536.doi:10.1109/ACCESS.2018.2812844.
3. Rui Guo et al. “Secure attribute-based signature scheme with multiple authorities for blockchain in electronic health records systems.” In:IEEE Access6 (2018),pp. 11676–11686
4. Cruz, Jason Paul, Yuichi Kaji, and Naoto Yanai. "RBAC-SC: Role-based access control using smart contract." *IEEE Access* 6 (2018): 12240-12251.
5. A. Azaria et al. “MedRec: Using Blockchain for Medical Data Access and Permission Management.” In:2016 2nd International Conference on Open and Big Data(OBD). Aug. 2016, pp. 25–30.doi:10.1109/OBD.2016.11.
6. Gavin Wood et al. “Ethereum: A secure decentralised generalised transaction ledger.” In:Ethereum project yellow paper 151.2014 (2014), pp. 1–32.
7. Rahat Masood, Muhammad Awais Shibli, Muhammad Bilal, et al. “Usage controlmodel specification in XACML policy language.” In:IFIP International Conference on Computer Information Systems and Industrial Management. Springer. 2012, pp. 68–79.
8. Jacob Eberhardt and Stefan Tai. “On or off the blockchain? Insights on off-chaining computation and data.” In:European Conference on Service-Oriented and Cloud Computing. Springer. 2017, pp. 3–15.
9. Quorum - Enterprise Ethereum Client.https://docs.goquorum.com/en/latest/. Accessed: 01/14/2020
10. Proof of Stake FAQ.https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ. Accessed: 01/09/2020
11. Raft-based consensus for Ethereum/Quorum.https://github.com/jpmorganchase/quorum/blob/master/docs/Consensus/raft.md. Accessed: 01/11/2020
12. "Testing Your Contracts".https://www .trufflesuite.com/docs/truffle/testing/testing-your-contracts. Accessed: 01/09/2020
13. Vitalik Buterin.Privacy on the Blockchain.https://blog.ethereum.org/2016/01/15/privacy-on-the-blockchain/. Accessed: 01/10/2020
14. Which OAuth 2.0 Flow Should I Use?https://auth0.com/docs/api-auth/which-oauth-flow-to-use. Accessed: 01/17/2020
15. Access Control.https://docs.openzeppelin.com/contracts/2.x/access-control, Accessed: 01/13/2020
16. Karl Wüst and Arthur Gervais. “Do you need a Blockchain?” In:2018 Crypto Valley Conference on Blockchain Technology (CVCBT). IEEE. 2018, pp. 45–54.
17. D. Di Francesco Maesa, P. Mori, and L. Ricci. “Blockchain Based Access ControlServices.” In:2018 IEEE International Conference on Internet of Things (iThings) andIEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical andSocial Computing (CPSCom) and IEEE Smart Data (SmartData). July 2018, pp. 1379–1386.doi:10.1109/Cybermatics_2018.2018.00237

B.Sc.

Thomas Hain

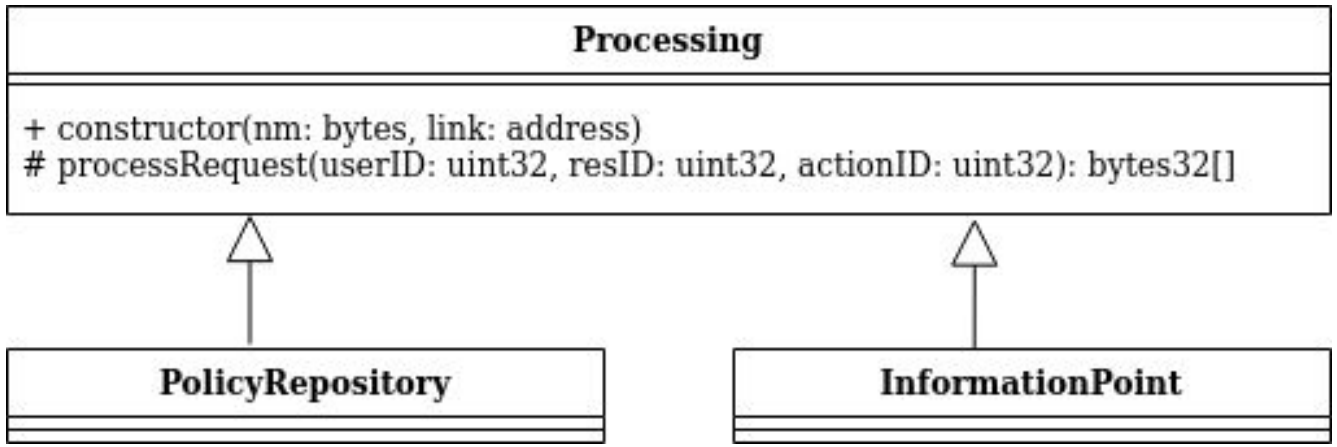
Technische Universität München
Faculty of Informatics
Chair of Software Engineering for Business Information Systems

Boltzmannstraße 3
85748 Garching bei München

ga46hom@mytum.de
wwwmatthes.in.tum.de

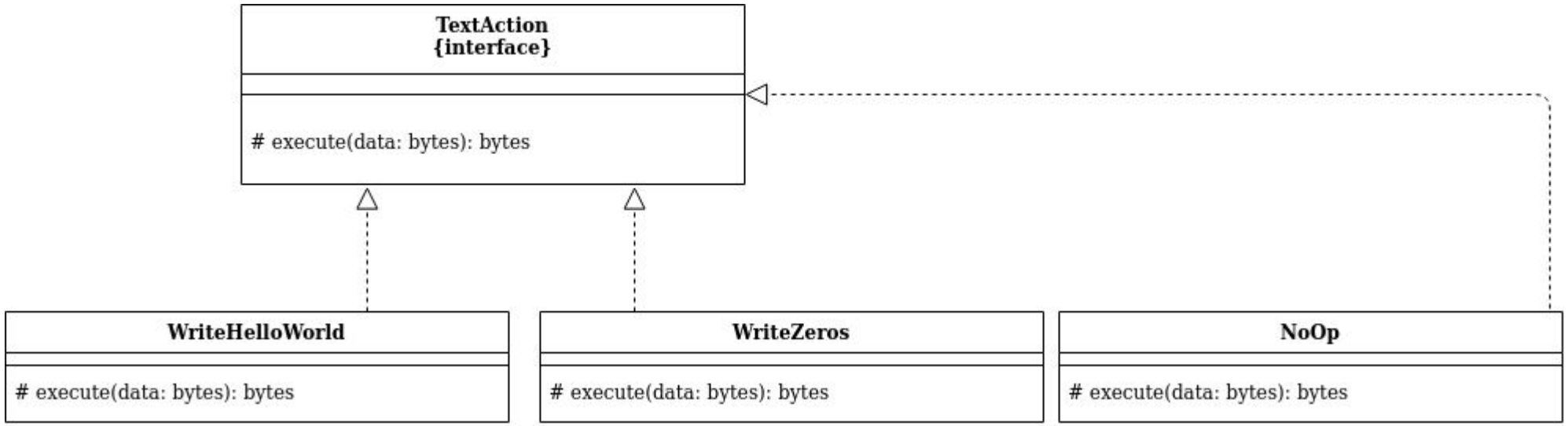
Backup - Processing

Processing



Backup - TextAction

TextAction



Takes argument of generic bytes
⇒ Returns bytes

Most basic form of action ⇒ More complex implementations are possible

Backup - OpenZeppelin

What are the advantages of OpenZeppelin's contracts?

+ Flexibility, e.g. via inheritance
+ Simple to use

Disadvantages:

- No policies
- Each protected function requires modifier annotation

Since OpenZeppelin is a framework its advantages underline its adaptability

Both contracts are rather basic

⇒ E.g. No separation of decision and enforcement

Most importantly

⇒ They don't accommodate any data privacy considerations

Backup - RBAC-SC

RBAC-SC

- Keeps a user array, Users are defined as structs
- Users have a string property “role”
- A modifier onlyOwner protects the functions to add and remove users

Similarity to OpenZeppelin’s RBAC.sol

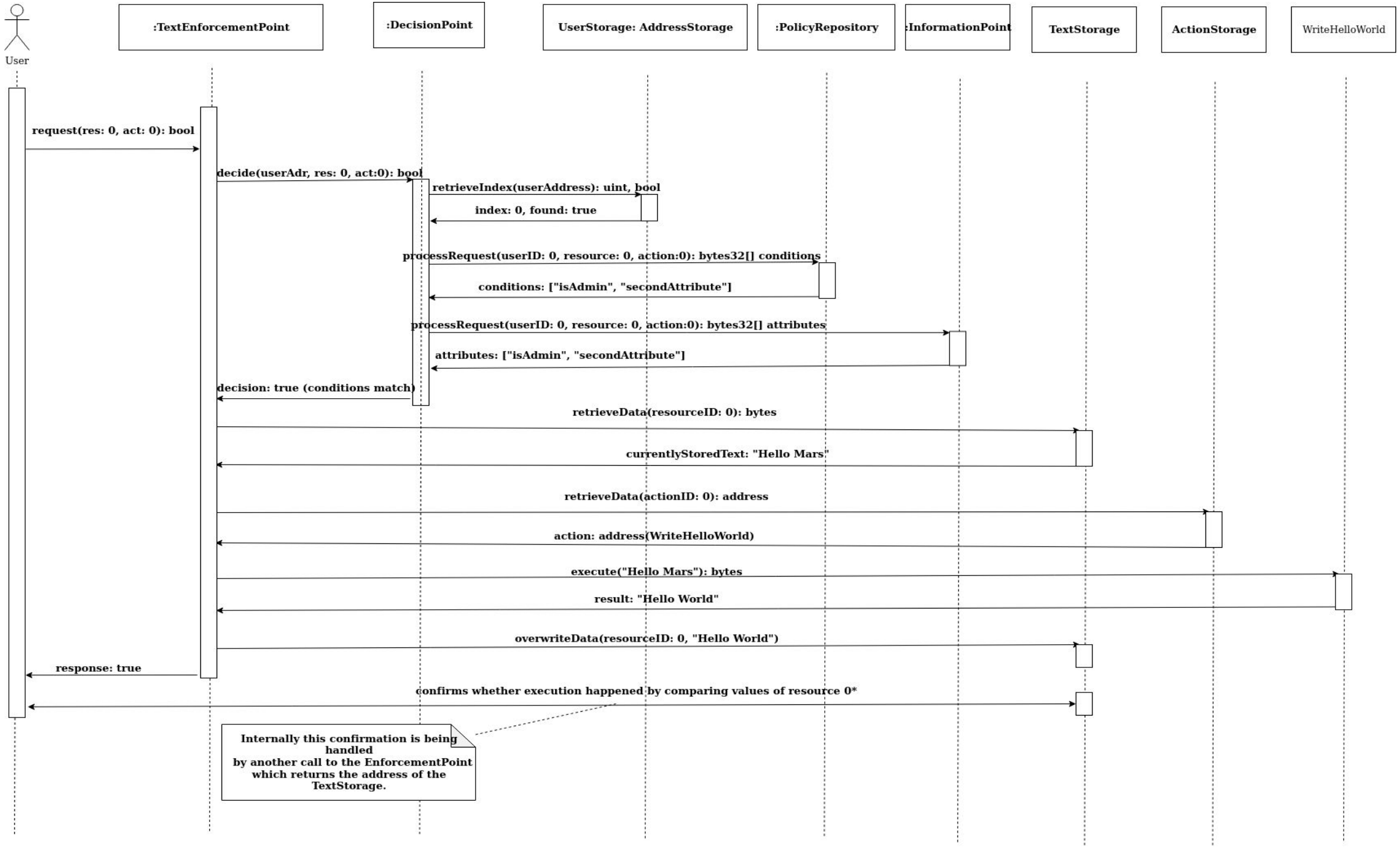
But it introduces no modifiers like “hasRole” etc.

Because:

- Enforcement happens off-chain
- User has to prove his role membership to another entity
- E.g. he owns the private key linked to the registered public key (his account)

Advantages & Disadvantages:

+	**Includes Privacy Considerations**	**-**	**No policies**
+	**Simple to implement**	**-**	**Only off-chain enforcement possible**


User
:TextEnforcementPoint
:DecisionPoint
UserStorage: AddressStorage
:PolicyRepository
:InformationPoint
TextStorage
ActionStorage
WriteHelloWorld
request(res: 0, act: 0): bool
decide(userAdr, res: 0, act:0): bool
retrieveIndex(userAddress): uint, bool
index: 0, found: true
processRequest(userID: 0, resource: 0, action:0): bytes32[] conditions
conditions: ["isAdmin", "secondAttribute"]
processRequest(userID: 0, resource: 0, action:0): bytes32[] attributes
attributes: ["isAdmin", "secondAttribute"]
decision: true (conditions match)
retrieveData(resourceID: 0): bytes
currentlyStoredText: "Hello Mars"
retrieveData(actionID: 0): address
action: address(WriteHelloWorld)
execute("Hello Mars"): bytes
result: "Hello World"
overwriteData(resourceID: 0, "Hello World")
response: true
confirms whether execution happened by comparing values of resource 0*
Internally this confirmation is being handled
by another call to the EnforcementPoint
which returns the address of the
TextStorage.

Backup - Full System Test

System Test with TextEnforcementPoint

- Randomly generated between 1 and 10 information Points
- Each of them provided between 1 and 20 different attributes
- The Decision Point's Policy Repository was added a single condition
- Test Case A) hid a fitting attribute within all the randomly generated attributes
  B) did not

Multiple iterations over both test cases

⇒ Assertion including fitting attribute corresponds to grant otherwise deny

Both scenarios were run 50 times

The Enforcement Point then acted accordingly

Backup - Tessera

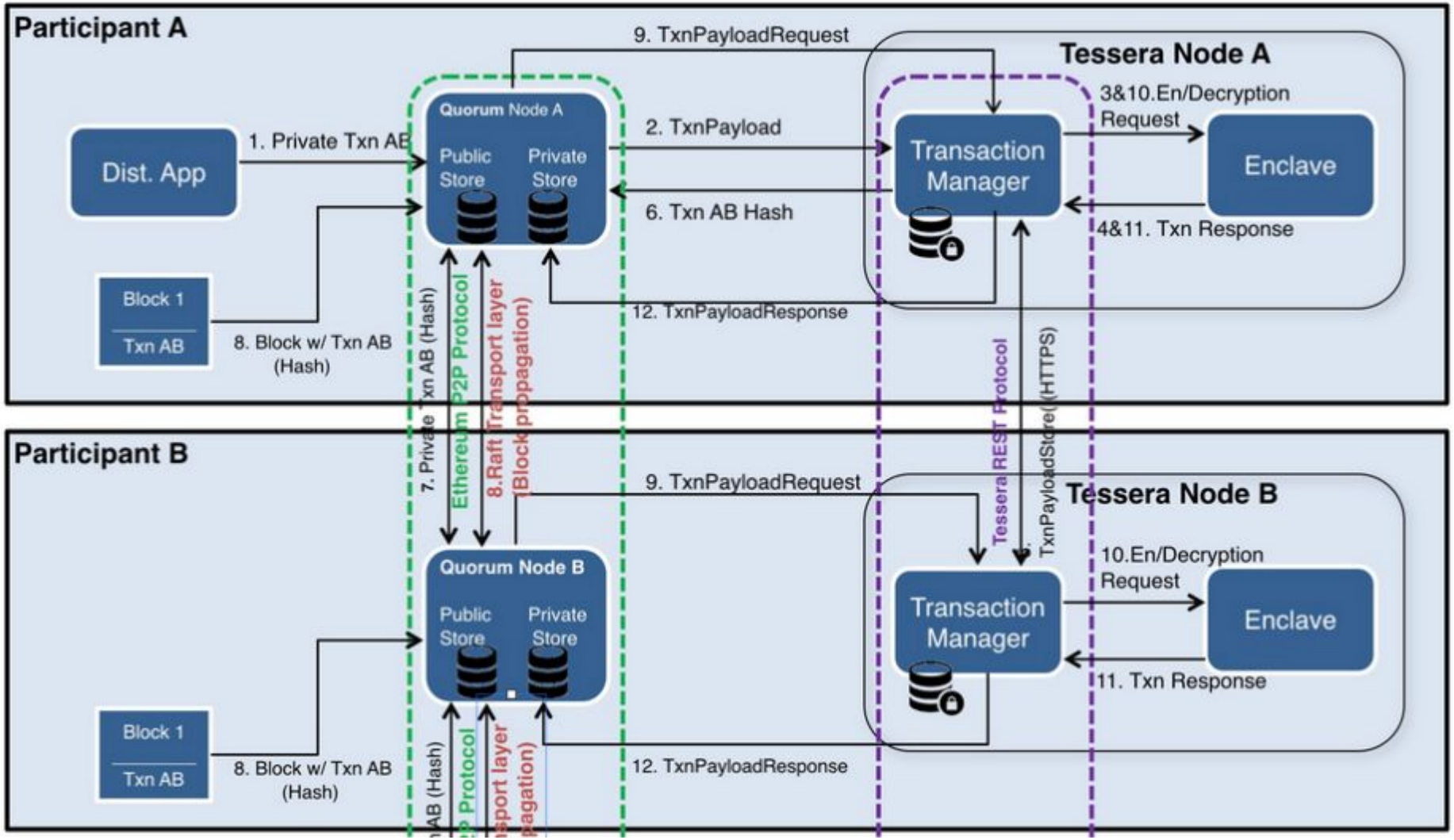
How is this achieved?

Quorum Introduces a Privacy Technology **“Tessera”**
Each Node is extended by

- **A Transaction Manager (TM)**
  ⇒ Passes private data to other nodes’ TMs **via HTTPS**
- **An Enclave**
  ⇒ Program encrypting and decrypting private transaction payloads
  ⇒ Only interacts with own Transaction Manager
  ⇒ Encrypts transaction payloads if necessary
  ⇒ Decrypts if transaction is intended to be read by current node

However using **private Smart Contracts** leads to loss of transparency
⇒ **breaks public verifiability**

Backup - Tessera



https://docs.goquorum.com/en/latest/Privacy/Tessera/How%20Tessera%20Works/

Backup - Advantages OpenZeppelin

What are the advantages of OpenZeppelin's contracts?

+ Flexibility, e.g. via inheritance
+ Simple to use

Disadvantages:

- No policies
- Each protected function requires modifier annotation

Since OpenZeppelin is a framework its advantages underline its adaptability

Both contracts are rather basic

⇒ E.g. No separation of decision and enforcement

Most importantly

⇒ They don't accommodate any data privacy considerations

Backup - Basic Authorization

Authorization can be **rather simple**

Example:

```
If (request.user == "admin"){
    //execute protected functionality
}
```

But it can consist of multiple different **"rules"** or **"policies"**
⇒ They can be structured hierarchically
⇒ Rules can potentially conflict each other

Policy framework is **XACML** ("eXtensible Access Control Markup Language")

It is being developed by the OASIS Consortium

⇒ It provides both an architecture as well as a policy language

Backup - Client-Server

Is there anything to learn from Client-Server?

Access Control by is also found within Client-Server systems

Traditionally it is subdivided into two parts

Authentication: Linking of user's identity with an internal representation (e.g. ID)

⇒ **Client-Server:** Database index mapping to hashed password

⇒ **Ethereum:** E.g. user array, mapping user's address to boolean values, ...

Authorization: What rights / permissions does the user have?

⇒ **Client-Server:** complex access control frameworks

⇒ **Ethereum:** often rather simple, patterns involve off-chain storage

Modern systems often don't rely on only a single server

⇒ Instead multiple connected API's

⇒ Authorization for all services can be handled by a single server (OAuth2)

Backup - Client-Server vs. Blockchain

Summarizing Requests on a Blockchain

- Users send public transactions to contracts
- Addresses can be used to store users
- Authorization either off-chain or rather simple because of gas consumption

In requests to REST-API's can be encrypted via HTTPS

⇒ They can contain sensitive data

But Smart Contract Interfaces are called via public transactions

So what is the current state of the art?

- How do RBAC-SC and OpenZeppelin implement Access Control?
- Do they approach Data Privacy or not?

Backup - OAuth2

OAuth2 is an Authorization Framework often used in Client-Server

It involves more than one app interacting with APIs.

Authorization Code Flow
Resource Owner (e.g. user)

Client (e.g. third party application)
Wants to interact with a user's resources

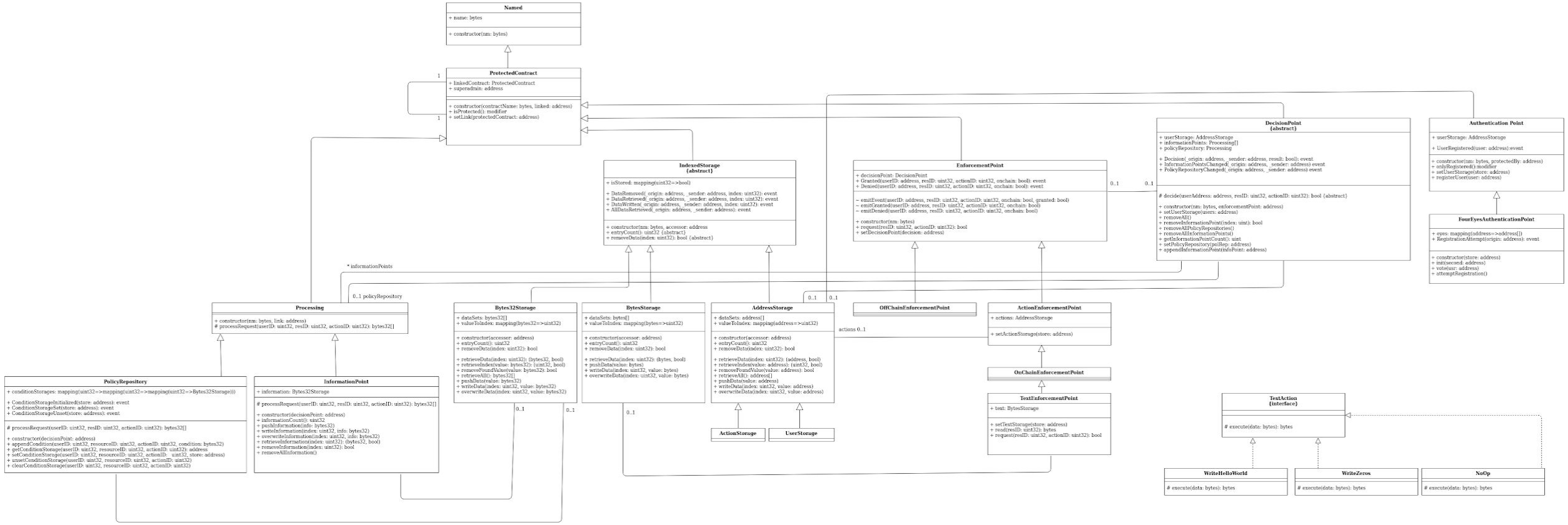
Resource Server (e.g. Google Drive)
- Interacts with both the Resource Owner and the Cilent
- ⇒ Client can request to resources via the Resource Server
- ⇒ The Resource Server prompts the user informing him about the request
- ⇒ If user grants access the Client receives an expiring access token

Backup - DecisionPoint

RequiredAttributesDecisionPoint

⇒ Every condition needs to be fulfilled by matching attribute
⇒ Maximum required condition threshold then grant by default
⇒ Otherwise deny

Backup - UML Gesamt



Backup - OpenZeppelin RBAC

https://docs.openzeppelin.org/docs/ownership_rbac_rbac

Modifiers

onlyRole

```
modifier onlyRole(string _role)
```

Modifier to scope access to a single role (uses msg.sender as addr).

Parameters:

`_role` - the name of the role // reverts

Functions

addRole

```
function addRole(address _operator, string _role) internal
```

Add a role to an address.

Parameters:

`_operator` - address

`_role` - the name of the role

Backup - OpenZeppelin

OpenZeppelin

Ownable.sol

- Contract contains a state variable “owner”
- It is initialized with the deployer’s address during its construction
- It includes a modifier “onlyOwner”

  ⇒ Exception before function call if a transaction’s sender is not the owner

Roles.sol

- Allows Role assignment to user addresses
- This is realized by chaining mappings
- A role is defined as a string
- Each role contains a mapping from type address => boolean to indicate membership
- Additionally it includes a modifier “onlyRole(role)” to protect functions