COMP6452 Lecture 8: Architectural Patterns for Blockchain Applications

Xiwei (Sherry) Xu (xiwei.xu@data61.csiro.au)

8th of April, 2019

Outline

- Design Pattern Essential
- What are Design Patterns?
- What are Architectural Patterns?
- Pattern Template

- Architectural Patterns for Blockchain-based Applications
- Overview
- Interaction with External World (5 patterns)
- Data Management (4 patterns)
- Security (4 patterns)
- Structural Patterns of Contract (5 patterns)
- Deployment (2 patterns)

- Summary

Design Pattern Essential

What is Design Pattern?

- To solve a recurring problem in software development

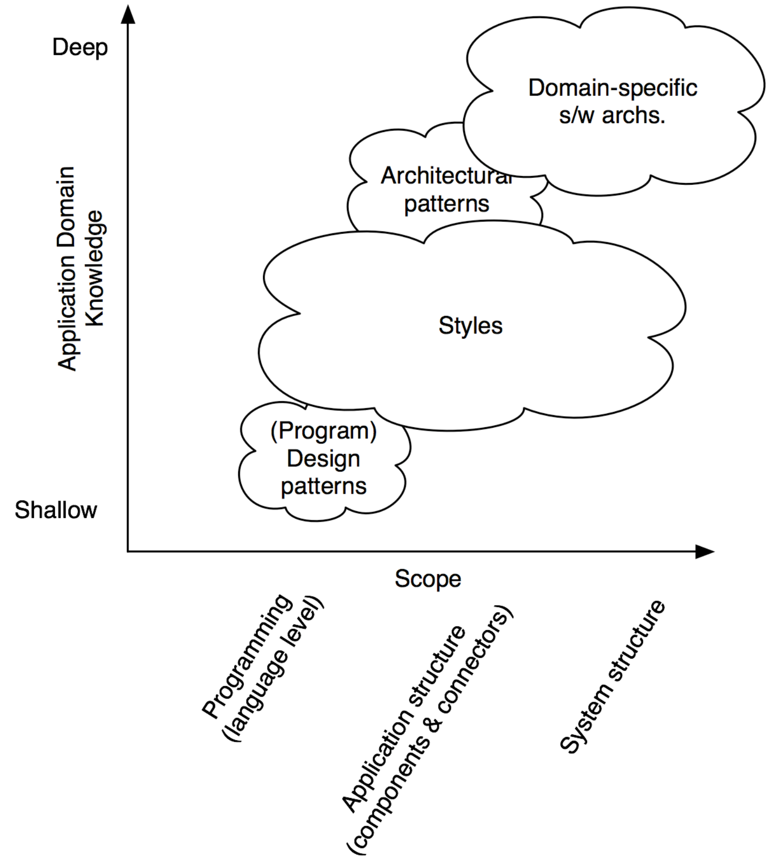
- Not a finished design that can be transformed directly into code

- A description or template for how to solve a problem
  - Define constraints that restrict the roles of architectural elements
    - Processing
    - Connectors
    - Data
  - Define constraints that restrict the interaction among these elements

- Cause trade-offs among quality attributes

Architecture Design

- From scratch
  - Unexpected solutions can be found
  - Labour-intensive and error-prone
- Apply a generic solution/strategy (Architectural style/ Design pattern) and adapt it to the problem
  - Reuse, less work and less errors
  - Generic solution might be ill-fitting or too generic



Software Architecture: Foundations, Theory, and Practice; Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy; © 2008 John Wiley & Sons, Inc.

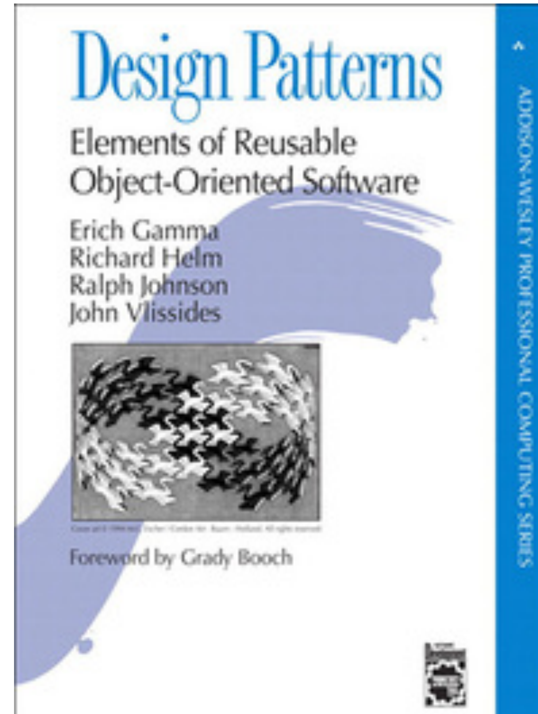
Advantages of Patterns

- Speed up the development process by providing tested, proven development paradigms
  - Design patterns document the efforts of the experts
  - Design patterns concern with a flexible software architecture
- Effective software design requires considering issues that may not become visible until later in the implementation
  - Reuse can prevent subtle issues that can cause major problems
  - No need to reinvent the wheel
- Better code readability for programmers and architects familiar with the patterns

Gang of Four (GoF)

Classic Object Oriented Design Patterns

- GoF are Erich Gamma, Richard Helm, Ralph Johnson and Jonh Vissides
- GoF document 23 classic software design patterns in their book
  - Design Patterns: Elements of Reusable Object-Oriented Software
- The GoF book published at October 1994 and documented design patterns already exist but not documented before



Four Essential Elements

Pattern Name	• Describe a design problem, its solutions and consequences in a word or two
Problem	• Explain the problem and its context • Conditions must be met
Solution	• Describe the elements that make up the design • Relationship, responsibilities, and collaborations
Consequence	• Results and trade-offs of applying the pattern • Critical for understanding the costs/benefits

Non-Functional Properties

Non-Functional Properties arise from Architectural Design Choices

- There are two kinds of requirements:

- Functional Requirements (i.e. what are the inputs and outputs)

- Non-Functional Requirements (a.k.a. *Qualities*, or *–illities*)
  - e.g. "Performance" (latency, throughput, …)
  - e.g. "Security" (confidentiality, integrity, availability, privacy, …)
  - e.g. Usability, Reliability, Modifiability, …

ISO/IEC 25010:2011 Quality Model

- **Functional suitability**
  - Functional correctness
  - Functional completeness
  - Functional appropriateness
- **Performance efficiency**
  - Capacity
  - Resource utilization
  - Time behavior
- **Compatibility**
  - Interoperability
  - Co-existence
- **Usability**
  - Operability
  - User error protection
- **Reliability**
  - Availability
  - Recoverability
  - Maturity
  - Fault tolerance
- **Security**
  - Integrity
  - Confidentiality
  - Non-repudiation
  - Accountability
  - Authenticity
- **Maintainability**
  - Modularity
  - Reuseability
  - Modifiability
  - Testability
  - Analyzability
- **Portability**
  - Installability
  - Replaceability
  - Adaptability

Adopting a design pattern causes trade-offs among quality attributes

Blockchain-based Application Pattern Collection

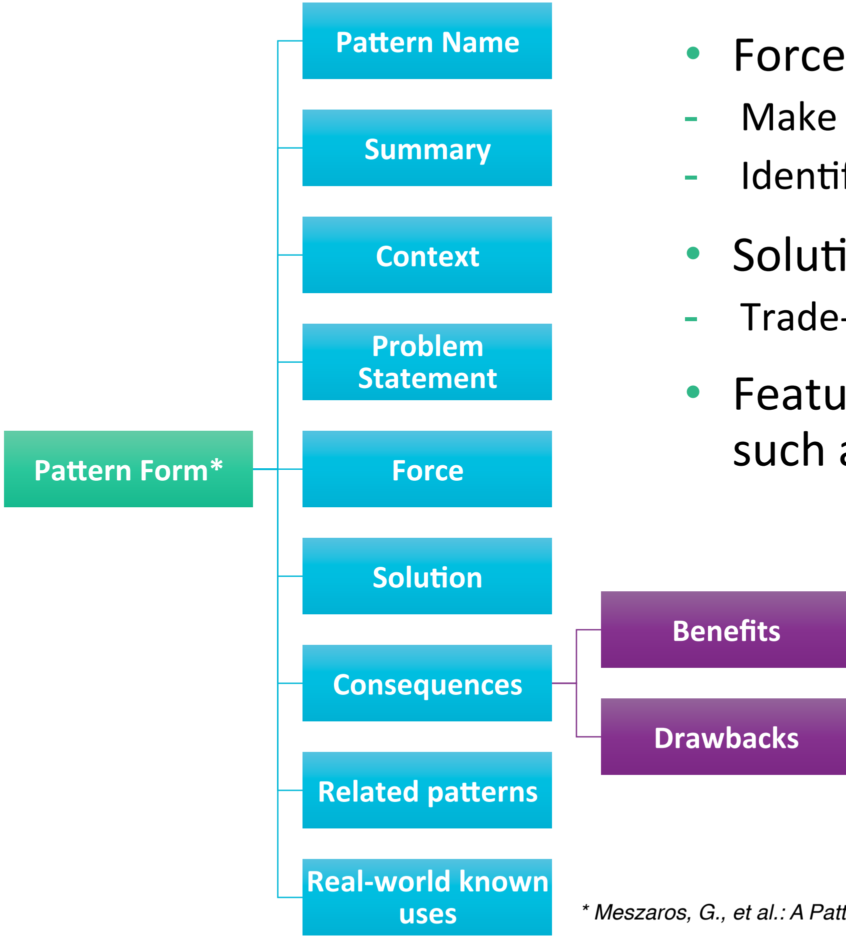
Pattern Collection

- **Interaction with External World**
  - Centralized Oracle
  - Decentralized Oracle
  - Voting
  - Legal and Smart Contract Pair
  - Reverse Oracle
- **Data Management**
  - Off-chain Data Storage
  - State Channel
  - Encrypting On-chain Data
  - Tokenisation
- **Security**
  - Dynamic Authorization
  - Multiple Authorization
  - X-confirmation
  - Security Deposit
- **Contract Structural Patterns**
  - Contract Registry
  - Embedded Permission
  - Data Contract
  - Factory Contract
  - Incentive Execution
- **Deployment**
  - Semi-dapp
  - dapp

Off-chain

On-chain

Pattern Form



- Forces
  - Make the problem difficult
  - Identified with the quality attributes
- Solution
  - Trade-off between the quality attributes
- Features only applicable to certain type of blockchain, such as monetary cost

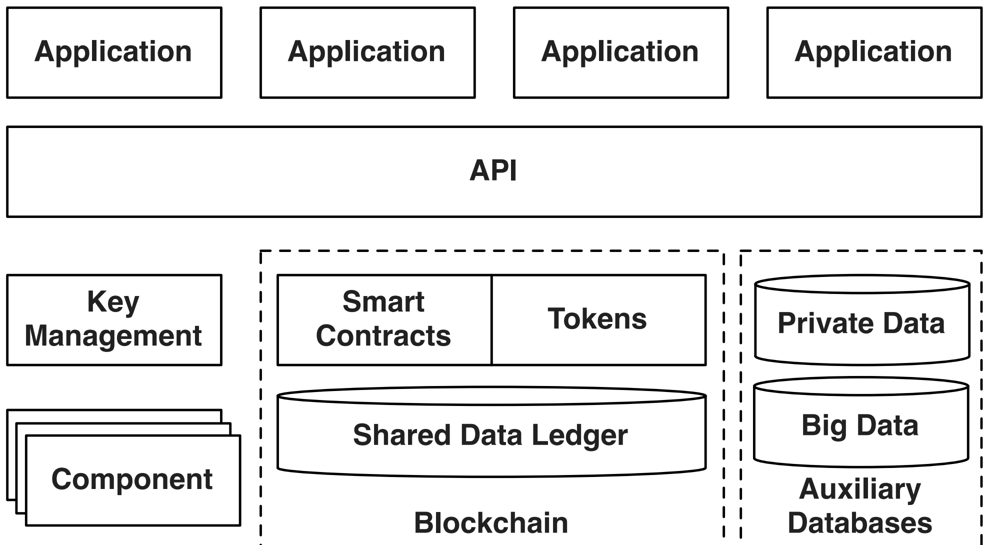
* *Meszaros, G., et al.: A Pattern Language for Pattern Writing. Pattern languages of program design (1998)*

Pattern Collection

- **Interaction with External World (5)**
- Data Management (4)
- Security (4)
- Contract (5)
- Deployment (2)

Overview

- Blockchain can be a component of a big software system
- Communicate with other components within the software system

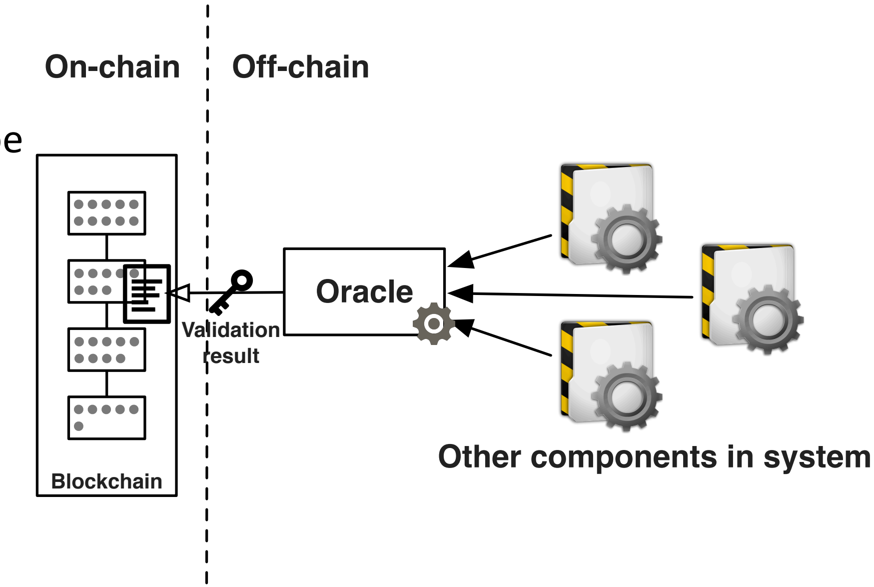


Pattern 1: Centralized Oracle 1/3

- **Summary**
  - Introduce the state of external systems into the closed blockchain execution environment through a single centralized oracle
- **Context**
  - Blockchain-based applications might need to interact with other external systems
  - Validation of transactions might depend on external state
- **Problem**
  - Blockchain is a self-contained execution environment
  - Smart contracts are pure functions that can't access external systems
- **Forces**
  - Closed environment
    - Secure, isolated execution environment
  - Connectivity
    - General-purpose applications might require information from external systems
  - Long-term availability and validity
    - External state used to validate a transaction may change or even disappear

Pattern 1: Centralized Oracle 2/3

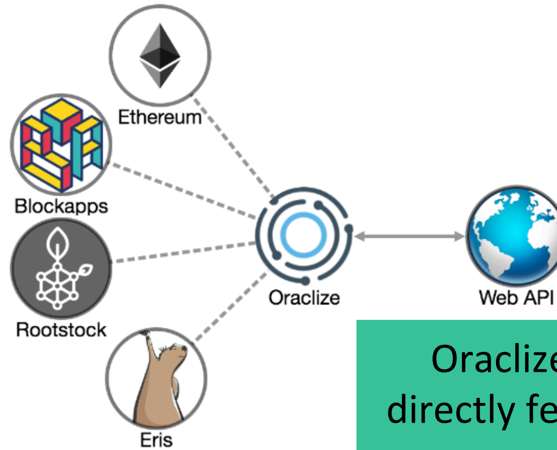
- **Solution**
  - Oracle assists in evaluating conditions that cannot be expressed in a smart contract
  - Oracle injects the result to the blockchain in a transaction signed using its own key pair
  - Validation of transactions is based on the authentication of the oracle



- **Consequences**
  - Benefits
    - Connectivity: Closed environment of blockchain is connected with external world through Oracle
  - Drawbacks
    - Trust: Oracle is trusted by all the participants
    - Validity: External states injected into the transactions can not be fully validated by miners
    - Long-term availability and validity: External state used to validate transaction changes after the transaction was originally appended to the blockchain

Pattern 1: Centralized Oracle 3/3

- **Related Patterns**
  - *Pattern 2. Decentralized Oracle*
  - *Pattern 4. Reverse Oracle*
- **Known uses**



Oraclize uses trusted hardware to directly fetch information from trusted execution environment (TEE)

Oracle in Bitcoin evaluates user-defined expressions based on the external state

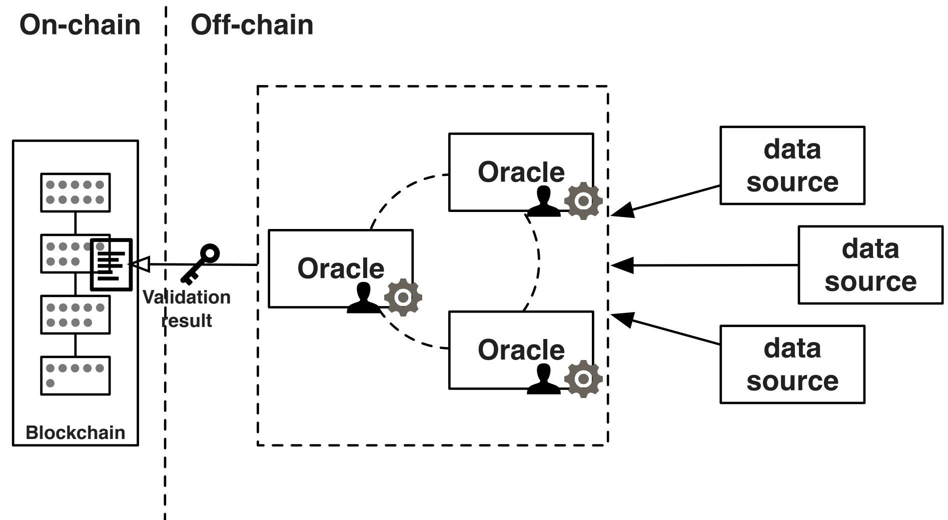
Corda has a embedded oracle mechanism using Intel Software Guard Extension (SGX) for hardware attestation to prevent unauthorized access outside of the SGX environment

Pattern 2: Decentralized Oracle 1/3

- **Summary**
  - Introduce the state of external systems into the closed blockchain execution environment through *decentralized* oracle
- **Context**
  - Blockchain-based applications might need to interact with other external systems
  - Validation of transactions relies on *oracle* to inject the external state
- **Problem**
  - A centralized oracle introduces a single trusted third party
- **Forces**
  - Reliability
    - Centralized oracle is a single point of failure
  - Variety of data sources
    - Static web page, physical sensor, input from a human
    - Multiple sources might come from a single authorized source

Pattern 2: Decentralized Oracle 2/3

- **Solution**
  - Decentralized oracle based on multiple servers and multiple data sources
  - Consensus on the external status
    - K-out-of-M threshold signature



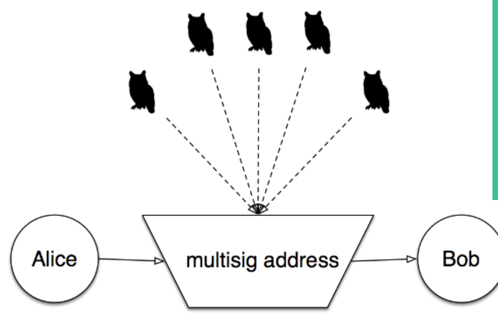
- **Consequences**
  - Benefits
    - Reliability
      - Risk is reduced from a single point of failure
      - Improves the likelihood of getting accurate external data
  - Drawbacks
    - Trust: All the oracles that verify the external state are trusted by all participants involved in transactions
    - Time: Get required information from multiple data sources and reach a consensus for the final result
    - Cost: increase with the number of oracles being used

Pattern 2: Decentralized Oracle 3/3

- **Related Patterns**

- *Pattern 1. Centralized Oracle*
- *Pattern 3. Voting*
- *Pattern 4. Reverse Oracle*

- **Known uses**



Orisi on Bitcoin allows participants involved in a transaction to select a set of independent oracles

augur

Augur is a prediction market that use human oracles

GNOSIS

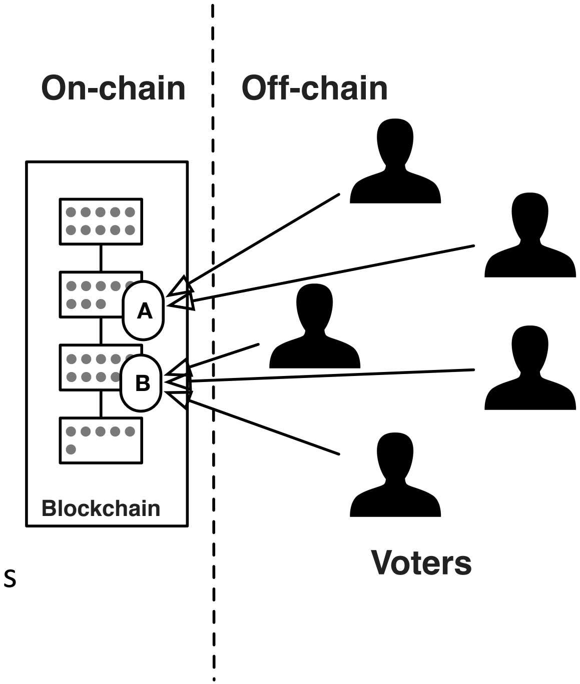
Gnosis is a prediction market allows users to choose oracles they trust

Pattern 3: Voting 1/3

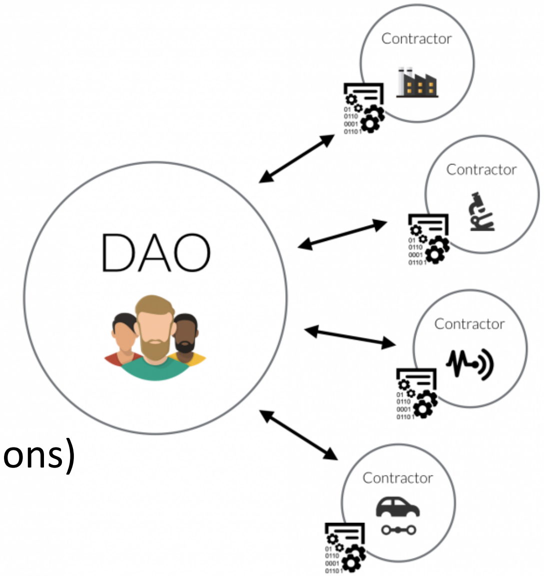
- **Summary**
  - Voting is a method for a group of blockchain users of a decentralized oracle to make a collective decision or to achieve a consensus
- **Context**
  - Public access of blockchain provides equal rights
  - Participant has the same ability to access and manipulate the blockchain
- **Problem**
  - Participants have different preference
- **Forces**
  - Decentralization
    - Devolves responsibility and capability from a central location to all the participants
  - Consensus
    - Participants need to reach an agreement to make decision

Pattern 3: Voting 2/3

- **Solution**
  - Vote through sending transaction through blockchain account
  - Voting transaction is signed by the private key
    - Represent the right to make decision
    - Might be weighted by the owned resource
- **Consequences**
  - Benefits
    - Equality: Participants use their right to make decision
    - Consensus: participants with different preferences can reach consensus
  - Drawbacks
    - Collusion: Collude during voting to gain benefit
    - Permission grant: Pseudonomity allows participant to gain additional voting power
      - Through owning multiple blockchain addresses
    - Time : Long voting/dispute time window



Pattern 3: Voting 3/3



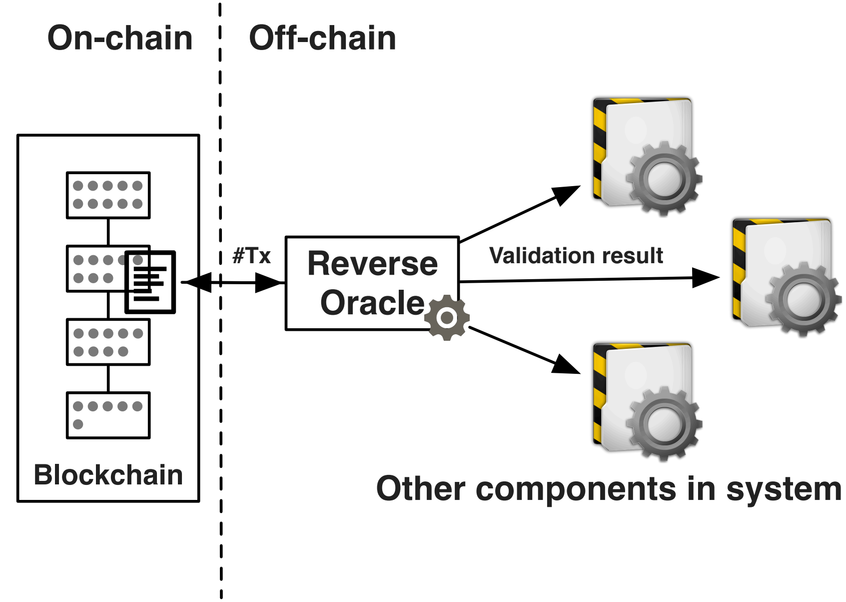
- **Related Patterns**
- *Pattern 2. Decentralized Oracle*
- *Pattern 10. Multiple Authorization*
- *Pattern 13. Security Deposit*

- **Known uses**
- Voting is used in DAOs (Decentralized Autonomous Organizations)
- Gnosis
  - Voting is used to challenge the reported outcomes
- Augur
  - Similarly, voting is used to resolve dispute

Pattern 4: Reverse Oracle 1/3

- **Summary**
  - The reverse oracle of an existing system relies on smart contracts to validate requested data and check required status
- **Context**
  - Off-chain components might need to use the data stored on the blockchain
  - Off-chain components might need to the smart contracts to check certain conditions
- **Problem**
  - Some domains use very large and mature systems, which comply with existing standards
  - Leverage the existing complex systems with blockchain without changing the core of the existing systems
- **Forces**
  - Connectivity
    - Integrate blockchain to leverage the unique properties of blockchain
  - Simplicity
    - Introduce minimal changes to the existing system

Pattern 4: Reverse Oracle 2/3



- **Solution**
  - Transaction ID and Block ID can be integrated into existing system
  - Validation is on blockchain using smart contract
  - An off-chain component is required to query blockchain

- **Consequences**
  - Benefits
    - Connectivity:
      - Blockchain is integrated into a system through adding IDs of transaction as a piece of data into the system
  - Drawbacks
    - Non-intrusive
      - Writing and reading blockchain might need changes to the existing system

Pattern 4: Reverse Oracle 3/3

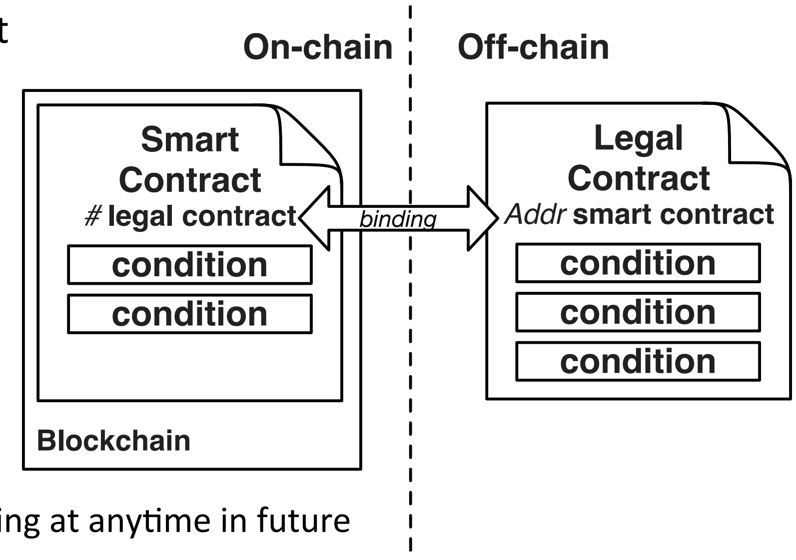
- **Related Patterns**
  - *Pattern 1. Centralized Oracle*
  - *Pattern 2. Decentralized Oracle*
- **Known uses**
  - Identitii
    - Enrich payment in banking systems with documents and attributes using blockchain
    - Identity token exchanged between the banks through SWIFT protocol
  - Slock.it
    - Autonomous objects and universal sharing network
      - Devices sell or rent themselves, and pay for services provided by others
    - Availability information is stored on blockchain
      - Validity checking is on blockchain

Pattern 5: Legal and Smart Contract Pair 1/3

- **Summary**
  - A bidirectional binding is established between a legal agreement and the corresponding smart contract
- **Context**
  - Legal industry is digitized
    - Digital signature is a valid way to sign legal agreements
  - Richardian contract (Mid 1990s)
    - interpret legal contracts digitally without losing the value of the legal prose
  - An independent trustworthy execution platform is needed to execute the digital legal agreement
- **Problem**
  - Bind a legal agreement to the corresponding smart contract to ensure 1-to-1 mapping
- **Forces**
  - Authoritative source: 1-to-1 mapping makes SC the authoritative source of legal contract
  - Secure storage: Blockchain is a trustworthy data storage
  - Secure execution: Blockchain provides a trustworthy computational platform

Pattern 5: Legal and Smart Contract Pair 2/3

- **Solution**
  - SC implements conditions defined in the legal agreement
    - Checked and enforced by the smart contract
  - SC has a blank variable to store hash of legal contract
  - SC address included in the legal agreement
  - Legal agreement hash is added to the SC variable



- **Consequences**
  - Benefits
    - Automation: SCs are programs running on blockchain
    - Audit trail: immutable historical transactions enable auditing at anytime in future
  - Drawbacks
    - Expressiveness: Smart contract language might have limited expressiveness to express contractual terms
    - Enforceability: No central authority to decide a dispute or perform the enforcement of a court judgment
    - Interpretation: Ambiguity of natural language is a challenge to accurately digitize a certain legal term

Pattern 5: Legal and Smart Contract Pair 3/3

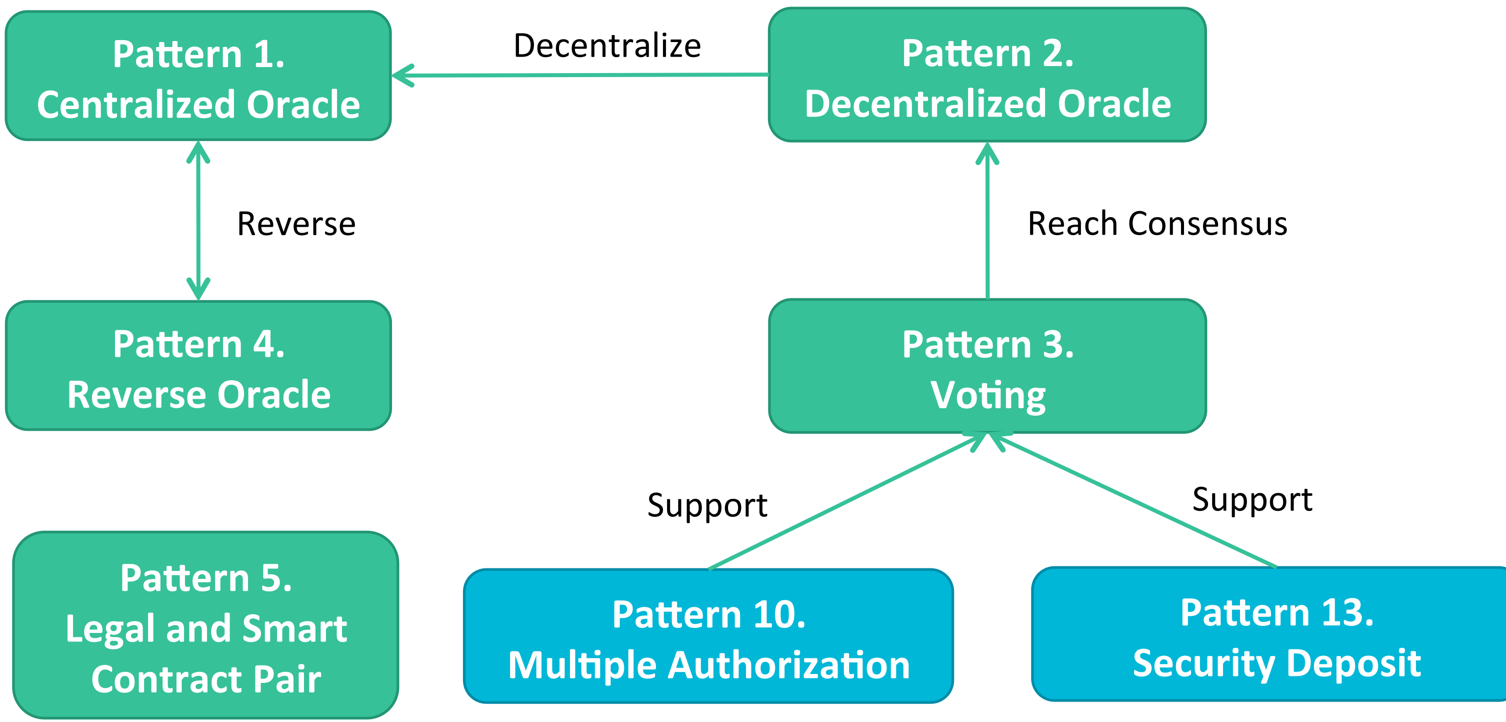
- **Related Patterns** N/A
- **Known uses**

- OpenLaw
  - Legally binding and self-executable agreements on the Ethereum blockchain
  - The legal agreement templates are stored on a decentralized data storage, IPFS

- Smart Contract Template proposed by Barclays uses legal document templates to facilitate smart contracts running on Corda

- Accord Project explored the representation of machine-interpretable legal terms

Related Patterns



Pattern Collection

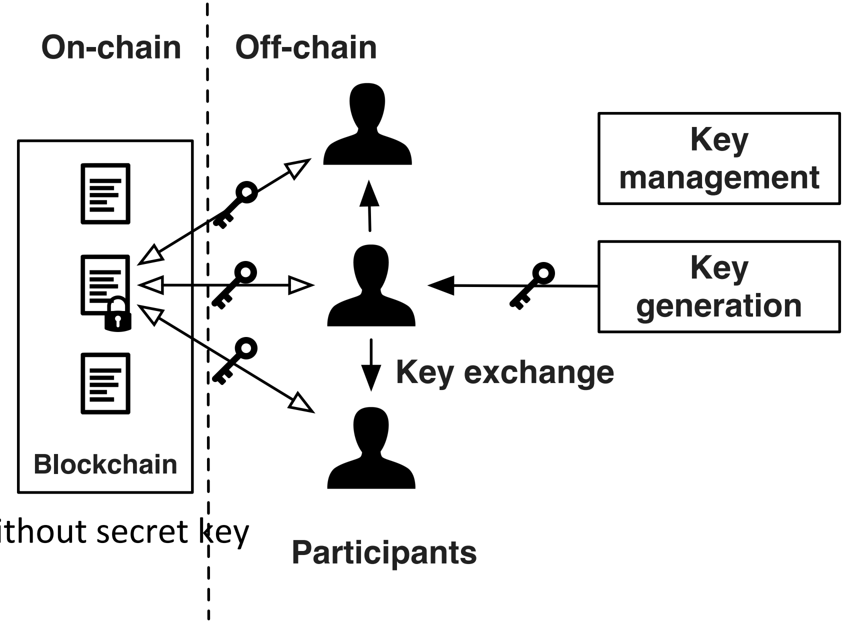
- Interaction with External World (5)
- **Data Management (4)**
- Security (4)
- Contract (5)
- Deployment (2)

Pattern 6: Encrypting On-chain Data 1/3

- **Summary**
  - Ensure confidentiality of the data stored on blockchain by encrypting it
- **Context**
  - Commercially critical data that is only accessible to the involved participants
    - Special discount price offered by a service provider to a subset of its users
- **Problem**
  - Data privacy is a limitation of blockchain
    - All information on blockchain is publicly available to all participants
    - No privileged user: no matter public/consortium/private blockchain
- **Forces**
  - Transparency
    - Historical transactions are publically accessible to enable validation of previous transactions
    - Transactions on public blockchain are accessible to anyone with access to internet
  - Lack of confidentiality
    - Commercially sensitive data should not be stored on blockchain

Pattern 6: Encrypting On-chain Data 2/3

- **Solution**
  - Data is encrypted before being inserted into blockchain
    - Symmetric encryption
    - Asymmetric encryption



- **Consequences**
  - Benefits
    - Confidentiality: Encrypted data is useless to anyone without secret key
  - Drawbacks
    - Compromised key: Encryption mechanism does not guarantee the confidentiality or integrity with a compromised or disclosed key
    - Access revocation: Access of Encrypted data is forever because of immutability
    - Immutable data: Subject to brute force decryption attack
      - Quantum computing
    - Key sharing: Off-chain key exchange otherwise accessible to all blockchain participants

Pattern 6: Encrypting On-chain Data 3/3

- **Related Patterns**
  - *Pattern 8. Off-Chain Data Storage*
- **Known uses**
  - Encrypted queries from Oraclize
    - Developers encrypt the parameters of their queries before passing them to a smart contract
    - Oraclize can decrypt the call parameters
  - Crypto digital signature from MLGBlockchain
    - Encrypting data before sharing data between the parties
  - Hawk* stores transactions as encrypted data on blockchain to retain the privacy of the transactions
    - Automatically generate a cryptographic protocol for a smart contract
    - Involved participants interact with the blockchain following the cryptographic protocol

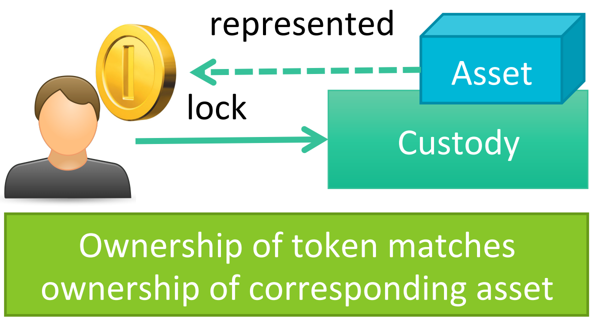
**Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: 37th IEEE Symposium on Security and Privacy (S&P2016)*

Pattern 7: Tokenization 1/3

- **Summary**
  - Using tokens to represent fungible goods for easier distribution
- **Context**
  - Reduce risk in handling high value financial instruments by replacing them with equivalents
    - Tokens used in casino
  - Tokens represent transferable and fungible goods
    - Shares or tickets
- **Problem**
  - Tokens representing assets should be the authoritative source of the corresponding assets
- **Forces**
  - Risk
    - Handling fungible financial instruments with high value is risky
  - Authority
    - Tokens are the authoritative source of the assets

Pattern 7: Tokenization 2/3

- **Solution**
  - Native tokens on blockchain used to represent digital or physical assets
    - Transactions record the verifiable title transfer from one user to another
    - With limited condition checking
  - Smart contract based data structure used to represent physical assets
  - On-chain token is the authoritative source of the physical asset



- **Consequences**
  - Benefits
    - Risk: Replacing high value financial instruments with equivalents
    - Authority
  - Drawbacks
    - Integrity: Authenticity of the physical asset is not guaranteed automatically
    - Legal process for ownership:
      - Owner of an asset may be entitled to sell the asset without being required to create a transaction on the blockchain

Pattern 7: Tokenization 3/3

- **Related Patterns** N/A

- **Known uses**

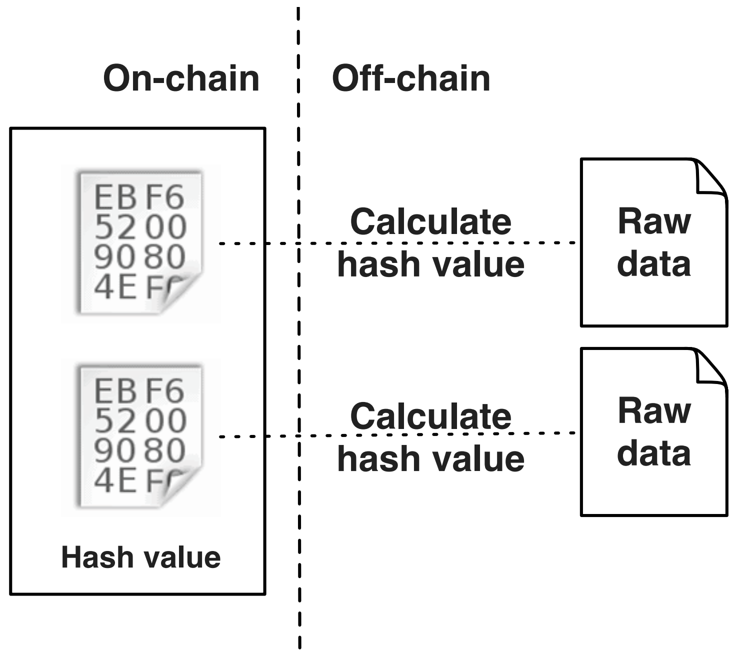
- Coloredcoin
  - Open source protocol for tokenizing digital assets on Bitcoin blockchain
- Digix
  - Use tokens to track the ownership of gold as a physical property

Pattern 8: Off-chain Data Storage 1/3

- **Summary**
  - Using hashing to ensure the integrity of arbitrarily large datasets
- **Context**
  - Using blockchain to guarantee the integrity of large amounts of data
- **Problem**
  - Limited storage capacity: full replication across all participants of the blockchain network
  - Limited size of the block: Storing large amounts of data within a transaction is impossible
    - Block gas limit in Ethereum
  - Data cannot take advantage of immutability or integrity guarantees without being stored on-chain
- **Forces**
  - Scalability: Data is replicated permanently across all nodes
  - Cost: Public blockchain charges real money: One-time cost
  - Size: Limits of transaction size or block size
    - Bitcoin relays *OP_RETURN* transactions up to 80 bytes

Pattern 8: Off-chain Data Storage 2/3

- **Solution**
  - Data of big size
    - Data that is bigger than its hash value
  - Hash value of the data is stored on blockchain
    - With other small sized metadata: a URI pointing to it
- **Consequences**
  - Benefits
    - Integrity:
      - Blockchain guarantees integrity of the hash value
      - Hash value guarantees integrity of the raw data
    - Cost: Fixed low cost for integrity of data with arbitrary size
  - Drawbacks
    - Integrity
      - Raw data might be changed without authorization
      - Detectable but unrecoverable
    - Data loss: Off-chain raw data may be deleted or lost



Pattern 8: Off-chain Data Storage 3/3

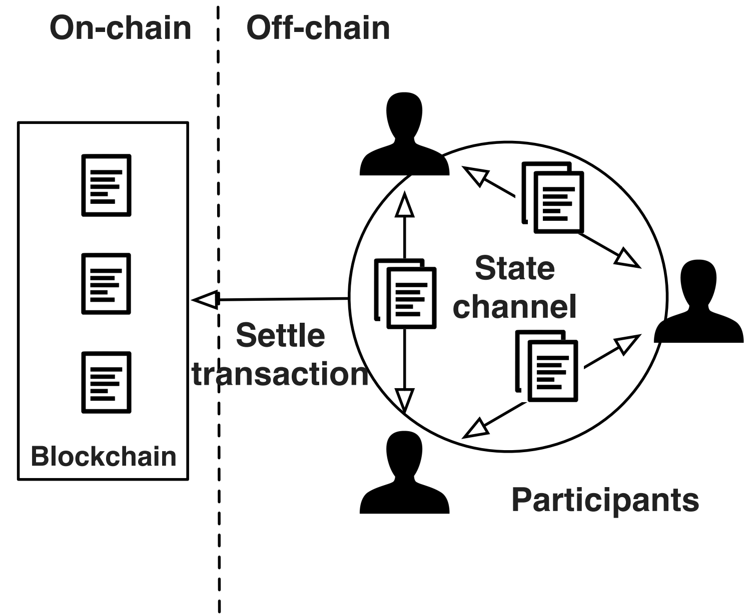
- **Related Patterns**
- *Pattern 9. State Channel*

- **Known uses**
- Proof-of-Existence (POEX.IO).
  - Entering an SHA-256 cryptographic hash of a document into the Bitcoin blockchain
  - A “proof- of-existence” of the document at a certain time
- Chainy
  - Smart contract running on Ethereum blockchain
  - Stores a short link to an off-chain file and its corresponding hash value in one place

Pattern 9: State Channel 1/3

- **Summary**
  - Micro-payments exchanged off-chain and periodically recording settlements for larger amounts on chain
  - Can be generalized for arbitrary state updates
- **Context**
  - Micro-payments are payments that can be as small as a few cents
    - E.g., payment of a very small amount of money to a WiFi hot-spot for every 10 KB of data usage
- **Problem**
  - Decentralized design has limited performance: Long commit time
  - High transaction fees on a public blockchain: Largely independent of the transacted amount
- **Forces**
  - Latency
    - Long commitment time on blockchain
    - Micro-payment is expected to happen instantaneously
  - Scalability: Data replicated permanently across all nodes
  - Cost : Transaction fee might be higher than the monetary value associated with micro-payment transaction

Pattern 9: State Channel 2/3



- **Solution**
  - Establish a payment channel between two participants
  - Deposit from one or both sides locked up
  - Payment channel keeps the intermediate states
  - Only the finalized payment is on chain
  - Frequency of settlement depends on use cases

- **Consequences**
  - Benefits
    - Speed: off-chain transaction settled without waiting for blockchain network to include the transaction
    - Throughput: off-chain transaction throughput is not limited by blockchain configuration
    - Privacy: intermediate off-chain transactions do not show up in the public ledger
    - Cost: only the final settlement transaction costs fee to be stored on blockchain
  - Drawbacks
    - Trustworthiness: Micro-payment transactions are not immutable and can be lost after the channel is closed
    - Reduced liquidity: Locked up security deposit reduces liquidity of channel participants

Pattern 9: State Channel 3/3

- **Related Patterns**

- *Pattern 8. Off-Chain Data Storage*

- **Known uses**

BITCOIN LIGHTNING NETWORK

- Hashed Timelock Contracts (HTLCs)
  - *Hashlocks* and *timelocks* of Script
  - Receiver acknowledges receiving the payment before deadline by proof
- Bi-directional payment channel

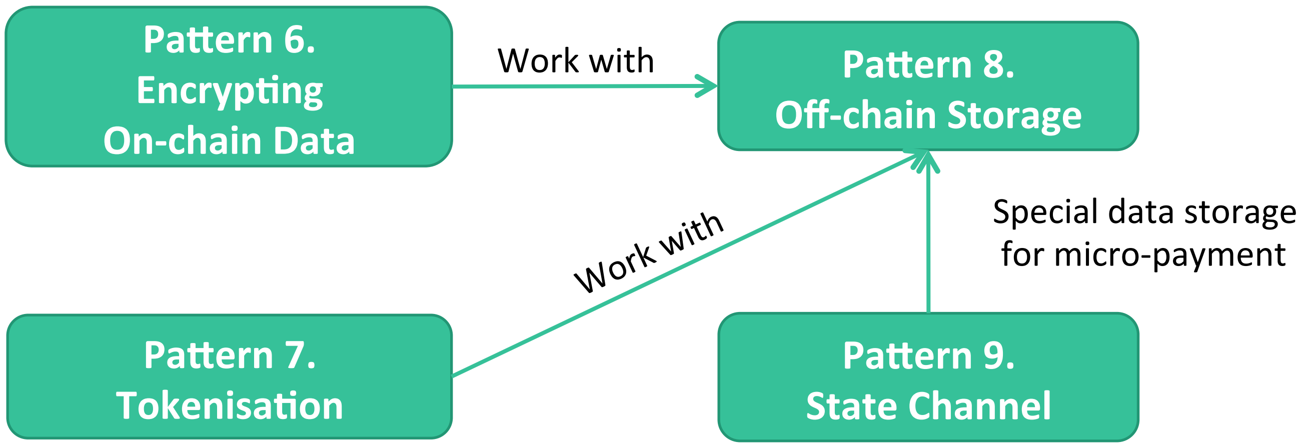
RAIDEN

Raiden network on Ethereum

Orinoco

Orinoco on Ethereum is a payment hub for payment channel management

Related Patterns



Pattern Collection

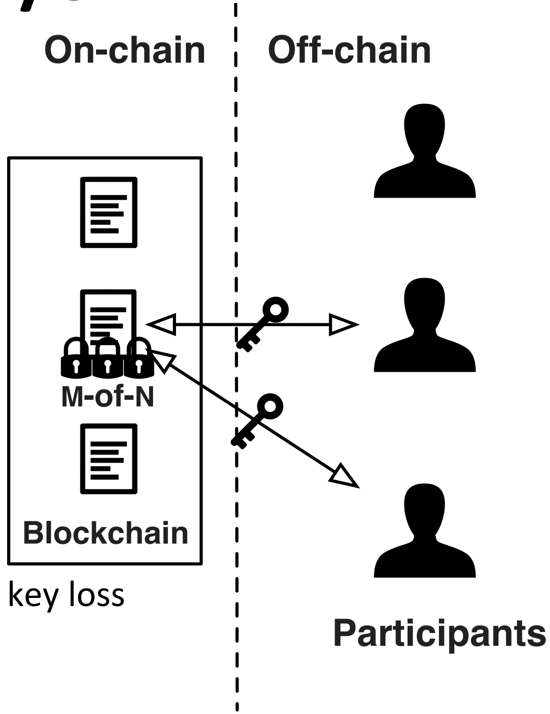
- Interaction with External World (5)
- Data Management (4)
- **Security (4)**
- Contract (5)
- Deployment (2)

Pattern 10: Multiple Authorization 1/3

- **Summary**
  - A set of blockchain addresses which can authorize a transaction is pre-defined
  - Only a subset of the addresses is required to authorize transactions
- **Context**
  - Activities might need to be authorized by multiple blockchain addresses
    - A monetary transaction may require authorization from multiple participants
- **Problem**
  - The actual addresses that authorize an activity might not be able to be decided due to availability
- **Forces**
  - Flexibility
    - The actual authorities can be from a set of pre-defined authorities
  - Tolerance of compromised or lost private key
    - Blockchain does not offer any mechanism to recover a lost or a compromised private key
    - Losing a key results in permanent loss of control over an account and smart contracts

Pattern 10: Multiple Authorization 2/3

- **Solution**
  - The set of blockchain addresses for authorization are not decided before the transaction being submitted to blockchain network
  - Multiple signature mechanism (M-of-N) is used to require more than one address to authorize a transaction



- **Consequences**
  - Benefits
    - Flexibility: Enable flexible binding of authorities based on availability
    - Lost key tolerance
      - Owning multiple addresses to reduce the risk of losing control due to key loss
      - Threshold-based authorized update
  - Drawbacks
    - Pre-defined authorities: All the possible authorities need to be known in advance
    - Lost key: At least M among N private keys should be safely kept to avoid losing control
    - Cost of dynamism: Extra logic and extra addresses cost extra money as does deploying the logic for multiple authorities

Pattern 10: Multiple Authorization 3/3

- **Related Patterns**

- *Pattern 3. Voting*
- *Pattern 11. Off-Chain Secret Enabled Dynamic Authorization*

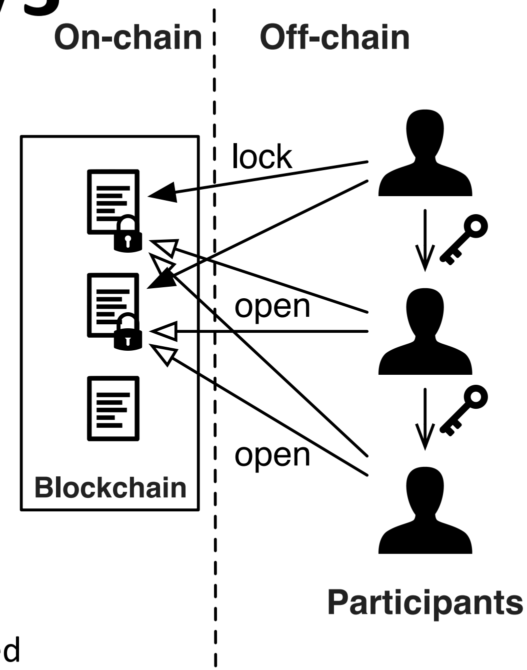
- **Known uses**

- MultiSignature mechanism provided by Bitcoin
- Multisignature wallet, written in Solidity, running on Ethereum blockchain
    - Available in the Ethereum DApp browser Mist

Pattern 11: Dynamic Authorization 1/3

- **Summary**
- Using a hash created off-chain to dynamically bind authority for a transaction
  - *Hashlock*
- **Context**
- Activities might need to be authorized by multiple blockchain addresses
  - These participants are unknown when a first transaction in submitted to blockchain
- **Problem**
- The authority who can authorize a given activity is unknown
- No dynamic binding with an address of a participant
  - All authorities for a second transaction are required to be defined in the first transaction
- **Forces**
- Dynamism: Dynamic binding multiple unknown authorities
- Pre-defined authorities: All the possible authorities are required to be defined beforehand if on-chain mechanism is used (Pattern 10)

Pattern 11: Dynamic Authorization 2/3



- **Solution**
  - Off-chain secret used to enable dynamic authorization
  - Transaction is "locked" by hash of an off-chain secret
    - E.g. a random string, called pre-image
  - Whoever receives the secret off-chain can authorize the transaction
- **Consequences**
  - Benefits
    - Dynamism: Enable dynamic binding of unknown authorities
    - Lost key tolerance: No specific key is required to authorize transaction
    - Routability: Enable multi-hop transfer since all payment transactions secured using the same secret can open at same time
    - Interoperability: Enable interaction between other systems and blockchain
  - Drawbacks
    - One-off secret: Secret is not reusable after being revealed
    - Lost secret: Transaction is "locked" forever if the secret is lost

Pattern 11: Dynamic Authorization 3/3

- **Related Patterns**
  - *Pattern 10. Multiple Authorization*

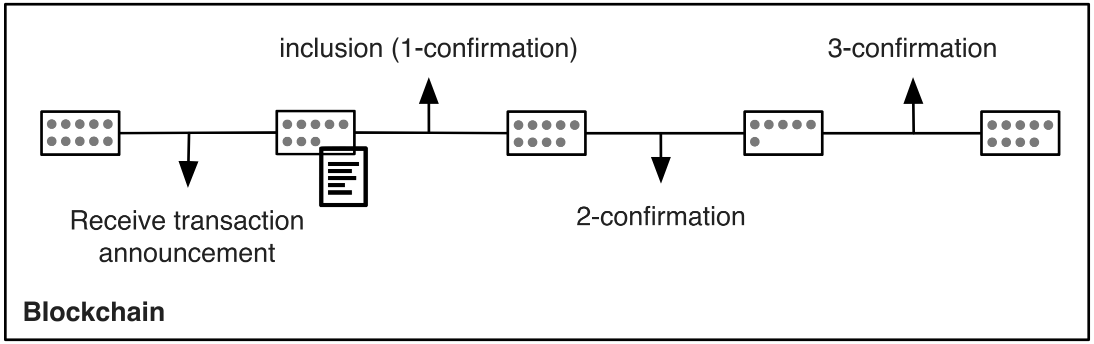
- **Known uses**
  - Raiden network
    - Multi-hop transfer mechanism
    - *hashlocked* transactions securely router payment through a middleman
  - Atomic cross-chain trading in the Bitcoin ecosystem
    - Trading Bitcoin for tokens on a Bitcoin sidechain

Pattern 12: X-Confirmation 1/3

- **Summary**
  - Waiting for enough number of blocks as confirmations to ensure that a transaction added into blockchain is immutable with high probability
- **Context**
  - Proof-of-work (Nakamoto) consensus enables probabilistic immutability
    - Most recent few blocks are replaced by a competing chain fork
    - Transactions included in the discarded branches go back to the transaction pool
- **Problem**
  - Fork: No certainty as to which branch will be permanently kept in blockchain
- **Forces**
  - Chain fork: Occurs on a blockchain using proof-of-work consensus
  - Frequency of chain fork
    - Shorter inter-block time would lead to an increased frequency of forks

Pattern 12: X-Confirmation 2/3

- **Solution**
  - Wait for a certain number (X) of blocks to be generated after the transaction is included into one block
  - Transaction is considered committed
  - X is blockchain specific



- **Consequences**
  - Benefits
    - Immutability: The more blocks generated after the block with the transaction, the higher probability of the immutability
  - Drawbacks
    - Latency
      - Latency between submission and confirmation of a transaction is affected by consensus protocol and X
      - The larger value of X, the longer latency

Pattern 12: X-Confirmation 3/3

- **Related Patterns** N/A

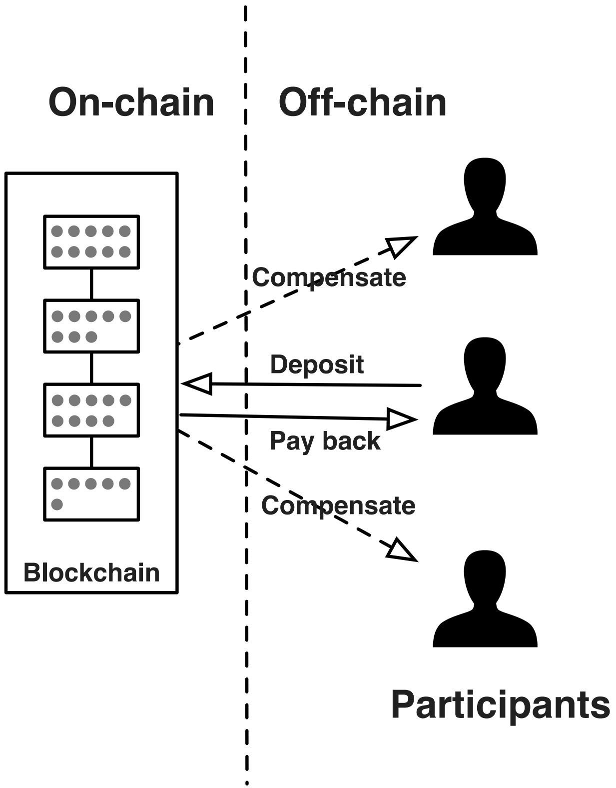
- **Known uses**
- Bitcoin uses 6-confirmation
  - An attacker is unlikely to amass more than 10% of the total amount of computing power
  - A negligible risk of less than 0.1% is acceptable
- Ethereum uses 12-confirmation

Pattern 13: Security Deposit 1/3

- **Summary**
  - A user put aside a certain amount of money, which will be paid back to the user for her honesty or given to the other parties to compensate them for the dishonesty of the user
- **Context**
  - Trust is achieved from the interactions between participants within the network
  - Blockchain-based applications relying on all the users to facilitate transactions
- **Problem**
  - Equal rights of blockchain allows every participant the same ability to manipulate the blockchain
  - How to prove honesty?
- **Forces**
  - Security: Security of the system relies on the behavior of all the participants
  - Incentive: Participants in a decentralized application can be incentivized to behave honestly

Pattern 13: Security Deposit 2/3

- **Solution**
  - Participant is required to put aside amount of tokens
    - Temporarily sacrificing stake
    - Recorded on blockchain
  - Paid back if the participant behaves honestly
  - Or compensate others for their lost due to dishonesty of the participant
- **Consequences**
  - Benefits
    - Security: Deposit is paid back only if the participant behaves honestly
      - Reduce the risk of participants misbehave
  - Drawbacks
    - Access
      - Security deposit is normally larger than the potential profit gain from dishonesty
      - Large security deposit restricts access to the application

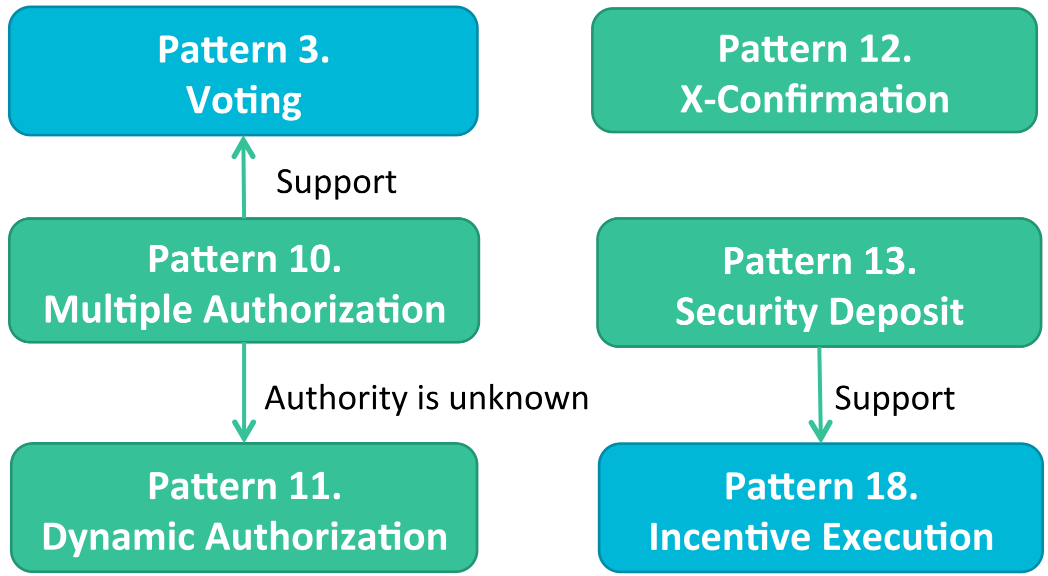


Pattern 13: Security Deposit 3/3

- **Related Patterns**
  - *Pattern 18. Incentive Execution*

- **Known uses**
  - Deposit used in Bitcoin contract
    - Party with no reputation buys deposit as a proof of trust
  - Slock.it requires servers to pay a deposit
    - Lost in case of a wrong response
    - Incentivize cross-checking as watchdog
  - Ethereum alarm clock enables scheduling of transactions for delayed execution in the future
    - *Claim window*: Deposit is required to claim a request
    - Return if the claimer fulfills the commitment to execute the request
    - Given to someone else that executes the request as an additional reward

Related Patterns



Pattern Collection

- Interaction with External World (5)
- Data Management (4)
- Security (4)
- **Contract (5)**
- Deployment (2)

Overview

- Smart contracts are programs running on blockchain
- Design patterns and programming principles for conventional software are applicable

- Structural design of the smart contract has large impact on its execution cost
- Monetary cost on public blockchain
- Cost of data storage proportional to the size of the smart contract
  - Applicable to both public and consortium blockchain

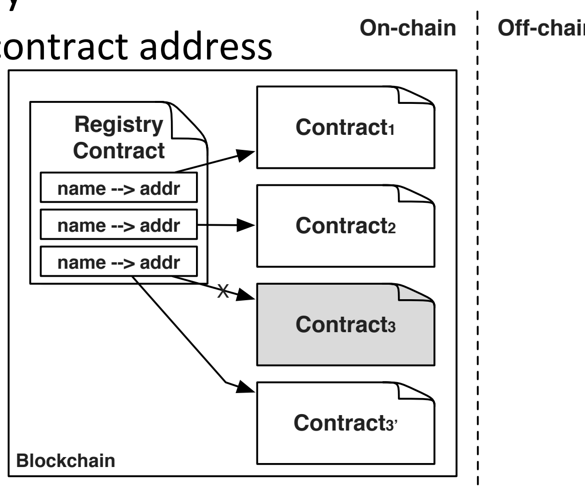
- Structural designs of smart contracts may affect performance
- More or less transactions may be required

Pattern 14: Contract Registry 1/3

- **Summary**
  - Before invoking it, the address of the latest version of a smart contract is located by looking up its name on a contract registry
- **Context**
  - Blockchain-based applications need to be upgraded to new versions
    - Fix bugs or fulfill new requirements
- **Problem**
  - Smart contract cannot be upgraded because of the immutable code stored on blockchain
- **Forces**
  - Immutability
    - On-chain contract code is immutable
  - Upgradability
    - Fundamental need to upgrade smart contract over time
  - Human-readable contract identifier
    - Hexadecimal address is not human-readable

Pattern 14: Contract Registry 2/3

- **Solution**
  - On-chain registry contract maintains a mapping between user-defined names and contract addresses
  - Registry contract address is advertised off-chain
  - Contract creator register the name and address of the new contract to the registry contract
  - Invoker retrieves the latest version of the contract from the registry
  - Upgrade contract by replacing the old contract address with new contract address



- **Consequences**
  - Benefits
    - Human-readable contract name
    - Constant contract name
    - Transparent upgradability
    - Version control: Look-up based on name and version
  - Drawbacks
    - Limited upgradability: Interface cannot be modified if the functions are called by others
    - Cost: Additional cost to maintain the registry and registry look-up for latest version

Pattern 14: Contract Registry 3/3

- **Related Patterns**
  - *Pattern 15. Data Contract*
  - *Pattern 16. Embedded Permission*

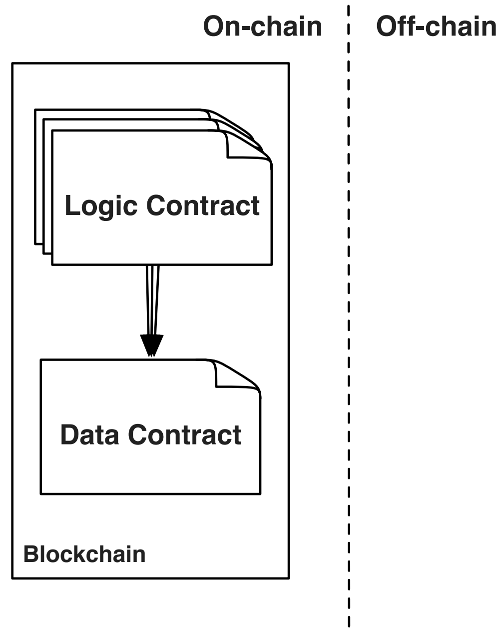
- **Known uses**
  - ENS (Ethereum Name Service)
    - Support registering both smart contract and off-chain resources
    - Contract registry accessible to everyone
      - Blockchain-based application can maintain a registry for the application
  - Regis
    - In-browser application for registries deployment and management
    - Allows user-defined key-value pairs for creating a contract registry

Pattern 15: Data Contract 1/3

- **Summary**
  - Store data in a separate smart contract
- **Context**
  - The need to upgrade application and smart contract over time is ultimately necessary
  - Logic and data change at different times and with different frequencies
- **Problem**
  - Upgrading transactions might need a large data storage for copying data from old to new contract
  - Porting data to new version might require multiple transactions
    - E.g. Ethereum gas limit prevents overly complex data migration transaction
- **Forces**
  - Coupling: The data stored in a deactivated contract is not accessible through SC functions
  - Upgradability: The need to upgrade application and smart contract is ultimately necessary
  - Cost : Copying data from old contract to new contract has extra cost

Pattern 15: Data Contract 2/3

- **Solution**
  - Data store is isolated from the rest of the code
    - Avoid moving data during upgrades of smart contracts
  - Strict definition or a loosely typed flat store
    - Depends on data store size and change frequency
  - More generic and flexible data structure can be used by other contracts
    - Less likely to require changes: Mapping between SHA3 key and value pair



- **Consequences**
  - Benefits
    - Upgradability: Application logic can be upgraded without affecting the data
    - Cost: No cost for migrating data when the logic is upgraded
    - Generality: Generic separated data contract can be used by multiple smart contracts
  - Drawbacks
    - Cost of generality: Generic data structure might cost more than a strictly defined data structure
    - Querying data with generic data structure is less straightforward

Pattern 15: Data Contract 3/3

- **Related Patterns**
  - *Pattern 14. Contract Registry*

- **Known uses**
  - Chronobank
    - Tokenize labor
    - Market for professionals to trade labor time with businesses
    - Generic data store uses a mapping of SHA3 key and value pairs

  - Colony
    - Ethereum-based platform for open organizations
    - Generic data store uses a mapping of SHA3 key and value pairs

Pattern 16: Embedded Permission 1/3

- **Summary**

- Smart contracts use an embedded permission control to restrict access to the invocation of their functions

- **Context**

- All the smart contracts can be accessed and called by all the participants and other smart contracts
- No privileged users

- **Problem**

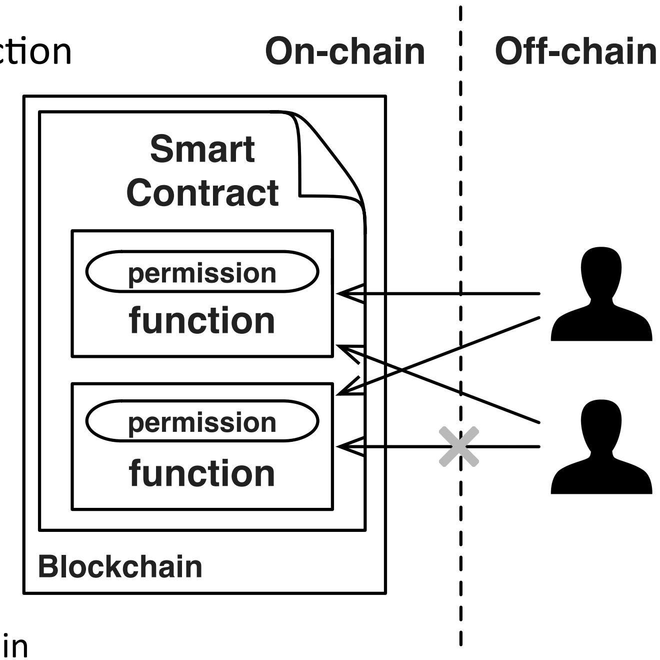
- Permission-less function can be triggered by unauthorized users accidentally
- Permission-less function becomes vulnerability
  - E.g., A permission-less function used by Parity multi-sig wallet caused freezing 500K Ethers

- **Forces**

- Security
  - Smart contract is publically available for everyone to invoke
  - Internal logic in conventional software system is normally invisible to end users
    - API/Interface is possible to enforce access control policies

Pattern 16: Embedded Permission 2/3

- **Solution**
  - Adding permission control before every smart contract function
    - Check authorization of the caller before executing the function
    - Check permission based on blockchain addresses
  - Unauthorized calls are rejected
- **Consequences**
  - Benefits
    - Security
      - Only authorized participants can successfully call functions
    - Secure authorization
      - Authorization leverages blockchain properties
  - Drawbacks
    - Cost
      - Additional deployment and run-time cost on public blockchain
    - Lack of flexibility
      - Permissions are defined before deployment and difficult to change
      - Mechanism needed to support dynamic granting and removal of permissions



Pattern 16: Embedded Permission 3/3

- **Related Patterns**
  - *Pattern 10. Multiple Authorization*
  - *Pattern 11. Dynamic Authorization*

```
contract owned {
    constructor() public { owner = msg.sender; }
    address owner;
}

contract mortal is owned {
    function kill() public {
        if (msg.sender == owner) selfdestruct(owner);
    }
}
```

```
modifier onlyBy(address _account)
{
    require(
        msg.sender == _account,
        "Sender not authorized."
    );
    // Do not forget the "_;"! It will
    // be replaced by the actual function
    // body when the modifier is used.
    _;
}

/// Make `_newOwner` the new owner of this
/// contract.
function changeOwner(address _newOwner)
    public
    onlyBy(owner)
{
    owner = _newOwner;
}
```

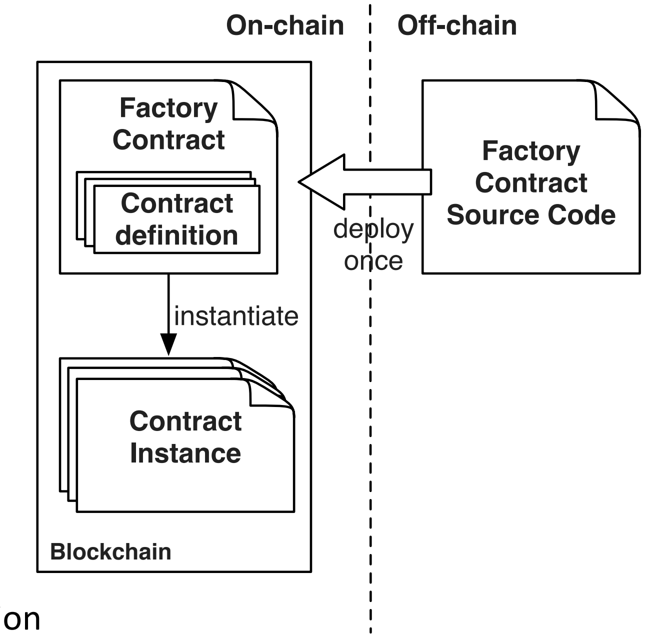
- **Known uses**
  - Mortal contract on Ethereum
    - Restricts the permission of invoking the self-destruct function to the contract owner
    - *Owner* is a variable defined in the contract
  - Restrict access pattern on Ethereum
    - Uses *modifier* to restrict who can call the functions
    - *Modifier* add a piece of code before the function to check certain conditions
      - Modifier makes such restrictions highly readable

Pattern 17: Factory Contract 1/3

- **Summary**
  - On-chain template contract is used as a factory that generates contract instances from the template
- **Context**
  - Application might need to use multiple instances of a standard contract with customization
  - Contract instance is created by instantiating a contract template
    - E.g. business process instances
  - Stored off-chain in a code repository or on-chain within its own smart contract
- **Problem**
  - Off-chain contract template cannot guarantee consistency between different SC instances
- **Forces**
  - Dependency management: Storing smart contract off-chain introduces more components
  - Secure code sharing: Blockchain is a secure platform for sharing contract code
  - Deployment: Extra effort is needed to deploy a smart contract from off-chain source cod

Pattern 17: Factory Contract 2/3

- **Solution**
  - Smart contract are created from a contract factory on blockchain
  - Factory contract is deployed once from off-chain source code
  - Smart contract instances are generated by passing parameters to the contract factory to instantiate customized instances
  - SC instance maintain its properties independently of the others
    - Class vs. Object



- **Consequences**
  - Benefits
    - Security: On-chain factory contract guarantees consistency
    - Efficiency: Smart contract instances are generated by calling a function
  - Drawbacks
    - Cost on public blockchain
      - Deployment
      - Function call for smart contract instance creation

Pattern 17: Factory Contract 3/3

- **Related Patterns**
  - *Pattern 14. Contract registry*

- **Known uses**
  - Tutorial from Ethereum developer
    - Create a contract factory
  - *Being applied in a real-world blockchain-based health care application
  - **A business process management system uses a contract factory to generate process instances

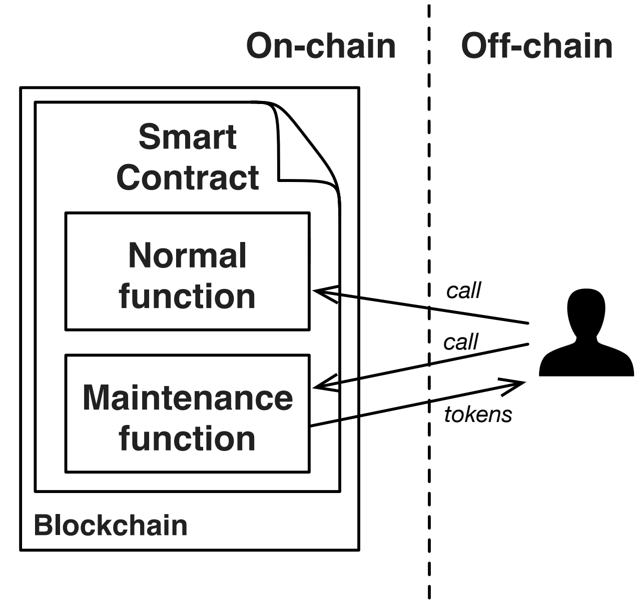
*Zhang, P., White, J., Schmidt, D.C., Lenz, G.: Applying Software Patterns to Address Interoperability in Blockchain-based Healthcare Apps (Jun 2017)
**Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., Mendling, J.: Untrusted business process monitoring and execution using blockchain. In: BPM. pp. 329–347. Springer, Rio de Janeiro, Brazil (Sep 2016)

Pattern 18: Incentive Execution 1/3

- **Summary**
  - Reward is provided to the caller of the contract function for invoking the execution
- **Context**
  - Smart contracts are event-driven
    - Cannot execute autonomously
  - Accessorial functions need to run asynchronously from regular user interaction
    - Clean up the expired records or make dividend payouts
    - Start after a time period
- **Problem**
  - Users have no direct benefit from calling accessorial functions
  - Extra monetary cost to call accessorial functions
- **Forces**
  - Completeness: Regular services are supported by accessorial functions
  - Cost: Execution of accessorial functions causes extra cost from users

Pattern 18: Incentive Execution 2/3

- **Solution**
  - Reward the caller of a function defined in a smart contract for invoking the execution
    - E.g., sending back a percentage of payout to the caller to reimburse execution cost
- **Consequences**
  - Benefits
    - Completeness: Execution of accessorial functions helps to complete the regular services
    - Cost: The caller is compensated by the reward associated with the execution
  - Drawbacks
    - Unguaranteed execution
      - Execution cannot be guaranteed even with incentive
      - Embed the logic of accessorial functions into other regular functions
        - Users have to call to use the services



Pattern 18: Incentive Execution 3/3

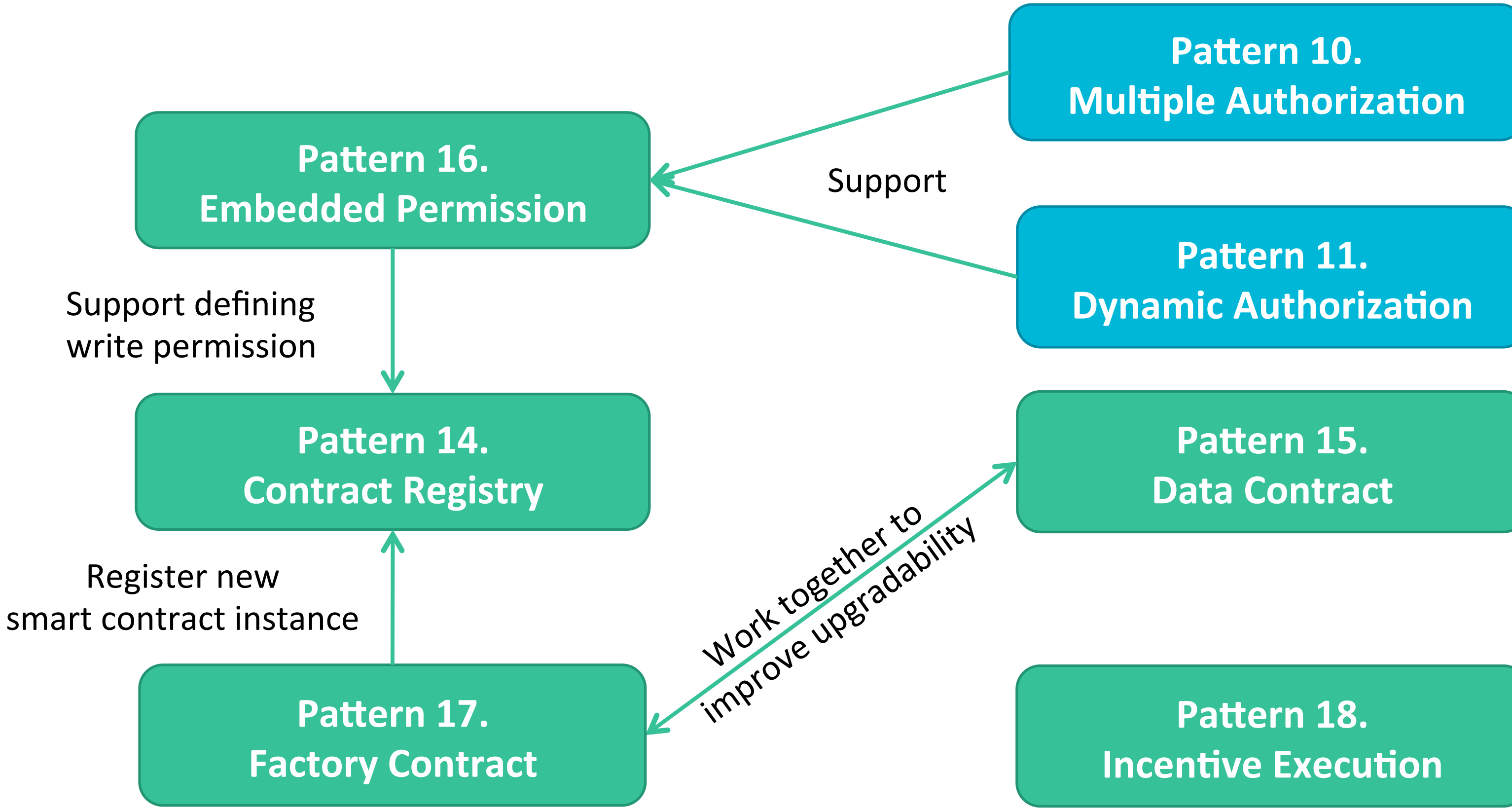
- **Related Patterns** N/A

- **Known uses**

- Regis
  - In-browser tool to create smart contract registries
  - Incentivize users to execute functions that clean up the expired records

- Ethereum alarm clock facilitates scheduling function calls for a specified block in the future
  - Provide incentive for users to execute the scheduled functions

Related Patterns



Pattern Collection

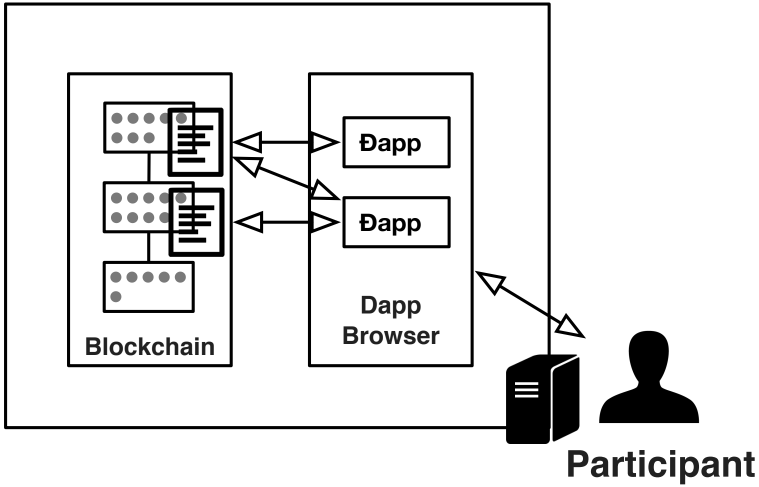
- Interaction with External World (5)
- Data Management (4)
- Security (4)
- Contract (5)
- **Deployment (2)**

Pattern 19: dapp 1/3

- **Summary**
  - Decentralized applications(dapps) are applications running on P2P network
  - Dapps are blockchain-based websites that allow users to interact with smart contracts
- **Context**
  - Users interacting with smart contracts through sending transactions to call smart contract
    - Source code of the smart contract should be open source
    - ABI (application binary interface) should be publicly accessible
- **Problem**
  - Strong technical understanding of blockchain and smart contract is required
  - Error-prone process with a bad user experience
- **Forces**
  - Learning Curve: Read source code →understand smart contract → interact with smart contract
  - Convenience: Manually generating transaction is a error-prone process

Pattern 19: dapp 2/3

- **Solution**
  - Front-end for users to interact with smart contract
  - Hosted on a decentralized storage
    - E.g., IPFS
  - Rendered by dapp browsers or plug-ins to web browser
    - E.g., Ethereum Mist or MetaMask
  - Transactions calling SC are generated by dapp
  - User verifies transactions before being sent to blockchain

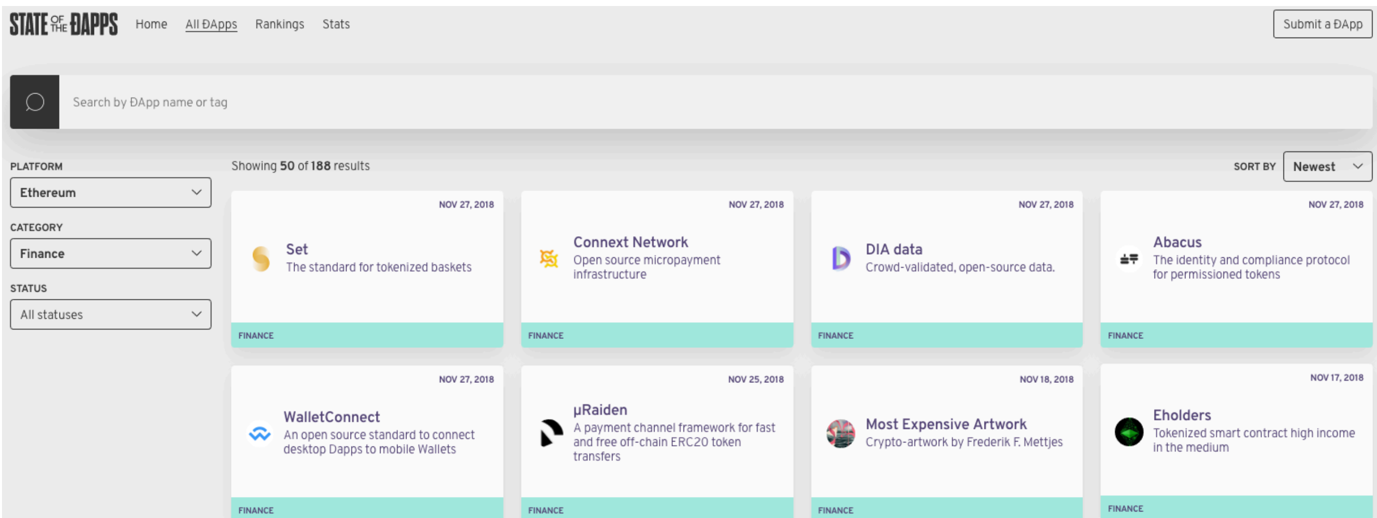


- **Consequences**
  - Benefits
    - Convenience: User experience of using front-end is much better than manually generating transactions
  - Drawbacks
    - Trust
      - Requires certain trust in the dapp provider
      - Impact of the transaction execution is not explicit without understanding the smart contract
    - Learning Curve: Require basic technical knowledge regarding transactions and smart contracts

Pattern 19: dapp 3/3

- **Related Patterns**
  - *Pattern 20. Semi-dapp*

- **Known uses**
  - *State of the dapp*
    - Directory of Dapps on Ethereum
    - 1800+ Dapps with different levels of maturity



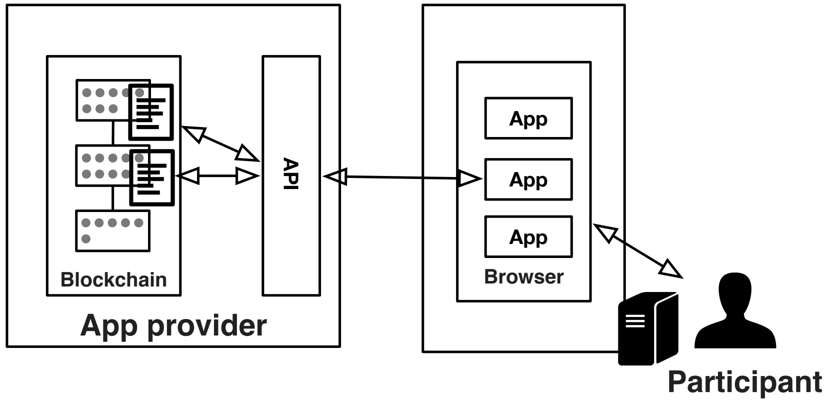
https://www.stateofthedapps.com/

Pattern 20: Semi-dapp 1/3

- **Summary**
  - dapp provider offers a website that can be browsed using a conventional web browser
- **Context**
  - Interacting with smart contracts through sending transactions to invoke execution
    - Source code of the smart contract should be open source
    - ABI (application binary interface) should be publicly accessible
- **Problem**
  - Even with a front-end assisting the user to interact with smart contracts
    - Basic knowledge regarding transactions and gas prices are required to use dapp and verify the transactions generated by dapp
- **Forces**
  - Learning Curve: Users need basic knowledge of smart contracts in order to use dapp
  - User Experience
    - Front-end of most dapps is quite simple
    - Exposes some technical detail of the underneath smart contract

Pattern 20: Semi-dapp 2/3

- **Solution**
  - Front-end for users to interact with smart contract
  - Rendered by Web browsers
    - Underneath SCs are invisible to the users
  - Website communicates with the dapp backend through RESTful API calls
  - Backend is responsible for interacting with the smart contracts on behalf of the user



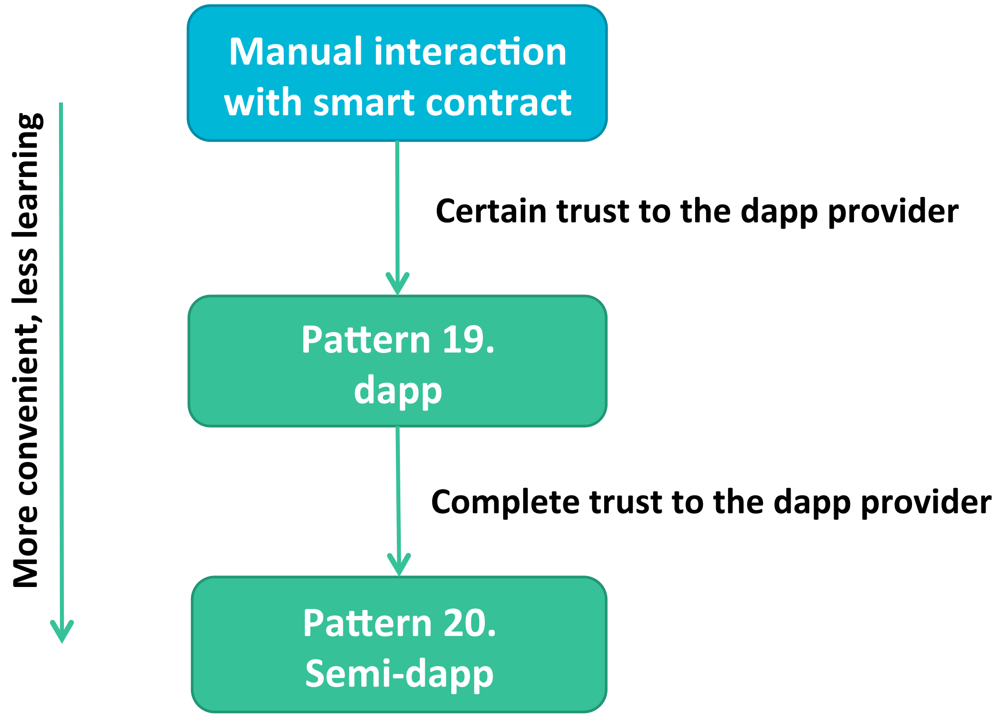
- **Consequences**
  - Benefits
    - Convenience: User experience is as same as conventional web applications
  - Drawbacks
    - Trust: Requires complete trust in the dapp provider
      - dapp provider manages the private keys of users
        - Mt. Gox lost 850,000 BTC due to a compromised internal computer
      - *Check the execution of the transactions sent by the dapp through an official blockchain exp...*

Pattern 20: Semi-dapp 3/3

- **Related Patterns**
  - *Pattern 19. dapp*

- **Known uses**
  - Cryptocurrency Exchanges
    - Kraken
      - Francisco-based Bitcoin exchange
    - Binance
      - China-based cryptocurrency exchange

Related Patterns



Blockchain-based Application Pattern Collection

Summary 1/2

Interaction with External World

Centralized Oracle

- Introducing external state into the blockchain environment through a centralized oracle

Decentralized Oracles

- Introducing external state into the blockchain environment through decentralized oracles

Voting

- A method for a group of blockchain users to make a collective decision

Reverse Oracle

- Reverse oracle relies on smart contracts to validate requested data and check status

Legal and smart contract pair

- A bidirectional binding between a legal agreement and the corresponding smart contract that codifies the legal agreement

Data Management

Encrypting On-chain Data

- Ensuring confidentiality of the data stored on blockchain by encrypting it

Tokenisation

- Using tokens on blockchain to represent transferable digital or physical assets or services

Off-chain Data Storage

- Using hashing to ensure the integrity of arbitrarily large datasets which may not fit directly on the blockchain

State Channel

- Transactions that are too small in value or that require much shorter latency, are performed off-chain with periodic recording of net transaction settlements on-chain

Security

Multiple Authorization

- Transactions are required to be authorized by a subset of the pre-defined addresses

Dynamic authorization

- Using a hash created off-chain to dynamically bind authority for a transaction

X-Confirmation

- Waiting for enough number of blocks as confirmation to ensure that a transaction added into blockchain is immutable with high probability

Security Deposit

- A deposit from a user, which will be paid back to the user for her honesty or given to others to compensate them for the dishonesty of the user

Summary 2/2

Structural Patterns of Contract

Contract Registry

- The address, and the version of the smart contract is stored in a contract registry

Embedded Permission

- Embedded permission control is used to restrict access to the invocation of the functions defined in the smart contracts

Data Contract

- Storing data in a separate smart contract

Factory Contract

- An on-chain template contract used as a factory that generates contract instances from the template

Incentive Execution

- A reward to the caller of a contract function for invocation

Deployment

dapp

- Blockchain-based application hosted on P2P network, with a website that allows users to interact with smart contracts

Semi-dapp

- Blockchain-based application with a website that can be browsed using a conventional web browser without any dapp plugin

Course Outline – Next two weeks

Week	Date	Lecturer	Lecture Topic	Relevant Book Chapters	Notes
8th	8 Apr	Sherry Xu	Design Patterns for Blockchain Applications	7. Blockchain Patterns	
9th	15 Apr	Ingo Weber	Model-Driven Engineering	8. Model-driven Engineering for Applications on Blockchain	Assignment 2 due on 17 Apr (Wednesday)
10th	22 Apr	Easter break			
11th	29 Apr	Mark Staples + Guest Lecturer	Guest Lecture + Summary		

THANK YOU

Xiwei Xu | Senior Research Scientist

Architecture & Analytics Platforms (AAP) team

t +61 2 9490 5664

e xiwei.xu@data61.csiro.au

w www.data61.csiro.au/

www.data61.csiro.au