Contract Architecture & Patterns

Technical Implementation Guide

Overview

This documentation outlines the implementation of a real-world asset tokenization system on Cardano, utilizing the Agora protocol for governance. The system enables the creation of fractionalized ownership of physical assets through secure smart contracts, with built-in governance mechanisms for collective decision-making.

The implementation combines vault management for asset handling, Agora's three-phase governance system, token policies for both fractional ownership and governance rights, and managed contribution windows. All components work together to create a secure, compliant, and flexible tokenization platform.

File Structure

```
plutus/
├── core/
│   ├── VaultFactory.hs        # Creates vault instances
│   ├── VaultRegistry.hs       # Global vault tracking
│   ├── FractionalizedVault.hs # Core vault functionality
│   └── VaultValidator.hs      # Main validation logic
├── agora/
│   ├── ProposalValidator.hs   # Proposal validation logic
│   ├── LockPhase.hs           # Lock phase implementation
│   ├── VotingPhase.hs         # Voting phase logic
│   ├── ExecutionPhase.hs      # Execution phase handling
│   └── ThresholdValidator.hs  # Governance thresholds
├── tokens/
│   ├── FractionalPolicy.hs    # Fractional token minting
│   └── GovernancePolicy.hs    # Agora governance token
├── windows/
│   ├── AssetWindowValidator.hs     # Asset contribution period
│   └── InvestmentWindowValidator.hs # Investment period
├── types/
│   ├── VaultTypes.hs          # Core type definitions
│   ├── AgoraTypes.hs          # Agora governance types
│   └── WindowTypes.hs         # Window related types
├── effects/
│   ├── ProposalEffects.hs     # Proposal action execution
│   └── RequirementEffects.hs  # Requirement modifications
```

```
└── utils/
    ├── Validators.hs        # Common validation functions
    └── Scripts.hs           # Script utilities
```

Core Vault Components

The vault system manages the fundamental asset tokenization functionality.

Vault Factory

Responsible for creating new vault instances with proper initialization:

```
data VaultParams = VaultParams {
    vaultType :: VaultType,
    assetTypes :: [AssetType],
    settings :: VaultSettings
}

newtype VaultFactory = VaultFactory {
    createVault :: VaultParams -> Contract w s Text VaultId
}

mkVaultValidator :: VaultParams -> TypedValidator VaultSchema
mkVaultValidator params = mkTypedValidator @VaultSchema
    ($$(PlutusTx.compile [|| validateVault ||])
        `PlutusTx.applyCode` PlutusTx.liftCode params)
    $$(PlutusTx.compile [|| wrap ||])
  where
    wrap = wrapValidator @VaultDatum @VaultRedeemer
```

Vault Registry

Maintains the global state of all vaults:

```
data RegistryDatum = RegistryDatum {
    vaults :: Map VaultId VaultInfo,
    totalVaults :: Integer
}

data VaultInfo = VaultInfo {
    vaultState :: VaultState,
```

```
    assetIds :: [AssetId],
    tokenPolicy :: CurrencySymbol
}
```

Fractionalized Vault

Manages the core vault operations and state:

```
data VaultDatum = VaultDatum {
    assets :: [AssetDetails],
    fractionalization :: FractionalizationParams,
    state :: VaultState
}

data AssetDetails = AssetDetails {
    assetId :: AssetId,
    amount :: Integer,
    locked :: Bool
}
```

Agora Governance Integration

The Agora protocol provides a structured governance system with three distinct phases.

Core Governance Structures

```
data GovernanceSettings = GovernanceSettings {
    creationThreshold :: Percentage,    -- Min FT % to create proposal
    startThreshold :: Percentage,       -- Min FT % to start voting
    voteThreshold :: Percentage,        -- Min staked FT for valid vote
    executionThreshold :: Percentage,   -- Min votes for execution
    cosigningThreshold :: Percentage,   -- Min FT for cosigning
    lockDuration :: POSIXTime          -- Lock phase duration
}

data ProposalPhase =
    LockPhase
  | VotingPhase
  | ExecutionPhase
```

Proposal Management

```
data ProposalParams = ProposalParams {
    description :: Text,
    votingDuration :: POSIXTime,
    lockDuration :: POSIXTime,
    executionDuration :: POSIXTime,
    requiredCosigners :: [PubKeyHash],
    effects :: [ProposalEffect]
}

createProposal :: VaultId -> ProposalParams -> Contract w s Text ProposalId
createProposal vaultId params = do
    validateCreatorStake
    proposalId <- submitTx $ mustPayToScript proposalValidator (proposalDatum params)
    emitEvent $ ProposalCreated proposalId
    pure proposalId
```

Phase Transitions

```
data PhaseTransition =
    StartVoting
  | StartLockPhase
  | StartExecution

validatePhaseTransition :: ProposalDatum -> PhaseTransition -> ScriptContext -> Bool
validatePhaseTransition datum transition ctx = case transition of
    StartVoting ->
        meetStartThreshold &&
        timeToVote
    StartLockPhase ->
        votingComplete &&
        sufficientVotes
    StartExecution ->
        lockPhaseComplete &&
        cosignersSigned
```

Token Management

Governance Token

```
data GovernanceToken = GovernanceToken {
    policyId :: CurrencySymbol,
    tokenName :: TokenName,
    totalSupply :: Integer
}

validateGovernanceToken :: AssetParams -> GovernanceToken -> ScriptContext -> Bool
validateGovernanceToken params token ctx =
    correctSupply &&
    correctDistribution &&
    hasGovernanceMetadata
```

Fractional Token

```
data FractionalTokenParams = FractionalTokenParams {
    tokenName :: TokenName,
    decimals :: Integer,
    totalSupply :: Integer
}

mkTokenPolicy :: FractionalTokenParams -> MintingPolicy
mkTokenPolicy params = mkMintingPolicyScript
    ($$(PlutusTx.compile [|| validateMinting ||])
        `PlutusTx.applyCode` PlutusTx.liftCode params)
```

Window Management

Asset Window

```
data WindowDatum = WindowDatum {
    windowStart :: POSIXTime,
    windowEnd :: POSIXTime,
    contributions :: Map AssetId Contribution
}

data WindowRedeemer =
    Contribute AssetContribution |
```

```
    CloseWindow
```

Types System

Vault Types

```
data VaultState =
    Draft
  | Active
  | Locked
  | Terminated

data VaultAction =
    Initialize VaultParams
  | AddAsset AssetDetails
  | RemoveAsset AssetId
  | LockAssets
  | UnlockAssets
```

Governance Types

```
data ProposalState =
    Pending
  | VotingActive
  | Locked
  | Executed
  | Failed

data ProposalAction =
    CreateProposal ProposalParams
  | CastVote VoteDetails
  | ExecuteProposal
  | CancelProposal
```

Implementation Notes

Best Practices

The implementation adheres to several key principles:

- Strong typing ensures that invalid states are unrepresentable in the type system. Every operation has explicit type definitions that capture its requirements and constraints.
- Explicit datum and redeemer structures clearly define the state transitions and valid operations. Each validator precisely specifies what constitutes a valid state change.
- The UTXO model is used efficiently, with careful consideration given to datum design and state management. This helps minimize resource usage and transaction costs.
- Validator patterns follow established security practices, with comprehensive checks and clear error messages. Multiple validation layers ensure system integrity.

Security Considerations

The implementation includes several security measures:

1. Multiple validation layers verify all operations
2. Strong typing prevents invalid state transitions
3. Explicit access control through stake-based governance
4. Comprehensive audit trail of all operations
5. Time-locked phases prevent rushed decisions

Integration Points

The system's main integration points are:

1. Asset registration and verification
2. Governance token distribution
3. Proposal creation and execution
4. Time Window management and transitions
5. Effect implementation and validation

Revision #5
Created 27 November 2024 16:15:18 by Aric Fedida
Updated 28 November 2024 00:49:30 by Aric Fedida