ブロックチェーンを用いた契約管理システムの試作

藤本 浩司[1]　井上 敦司[2]　三淵 啓自[1]

概要：従来の契約管理システムは第三者機関への依存度が高く，国際取引において仲介コストや手続きの煩雑化に伴うコスト増大が深刻化している．著者らは先行研究において，PoS（Proof of Stake）と DAO（自律分散組織）を組み合わせたトラストレスな契約管理システムを提案した．本研究では，この提案を具体化するため，PoA（Proof of Authority）コンセンサスアルゴリズムとカスタムステーキングを組み合わせたシステムの試作を行った．実装にあたっては，Avalanche Subnet を基盤としたモジュール型ブロックチェーンを構築し，Raspberry Pi 環境での実証実験を実施した．さらに，オブジェクト指向型スマートコントラクトにファクトリーパターンを導入することで，契約ロジックの再利用性と保守性を向上させた．実証実験の結果，PoA とカスタムステーキングにより「当事者=利用者」エコシステムが技術的に実現可能であること，オブジェクト指向設計により従来のモノリシック型と比較して保守性が大幅に向上すること，モジュール型アーキテクチャにより必要に応じたシステム拡張が可能であることを確認した．これにより，第三者機関に依存しない効率的な契約管理システムの実現可能性を実証した．

キーワード：契約管理システム，ブロックチェーン，スマートコントラクト，PoA，プライベートチェーン，DAO

Prototyping a Contract Management System Using Blockchain Technology

Hiroshi Fujimoto[1]　Atsushi Inoue[2]　Keiji Mitsubuchi[1]

Abstract: Conventional contract management systems rely heavily on third-party institutions, leading to a significant increase in costs from intermediation fees and procedural complexities in international transactions. In our previous research, we proposed a trustless contract management system combining PoS (Proof of Stake) and DAO (Decentralized Autonomous Organization). To realize this proposal, this study prototyped a system integrating the PoA (Proof of Authority) consensus algorithm with custom staking mechanisms. We constructed a modular blockchain based on an Avalanche Subnet and conducted proof-of-concept experiments in a Raspberry Pi environment. Furthermore, we improved the reusability and maintainability of the smart contracts by introducing the factory pattern into their object-oriented design. The experimental results confirmed the technical feasibility of a "stakeholder-as-user" ecosystem via PoA and custom staking. They also showed that the object-oriented design significantly improves maintainability over monolithic approaches and that the modular architecture enables flexible system expansion. These findings demonstrate the feasibility of an efficient contract management system that operates without depending on third-party institutions.

Keywords: contract management system, blockchain, smart contract, PoA, private chain, DAO

1. はじめに

1.1 研究背景

従来の契約管理システムが抱える第三者機関への依存度

[1] デジタルハリウッド大学大学院
Digital Hollywood University Graduate School
[2] 九州工業大学大学院
Kyutech Graduate School

の高さや，国際取引におけるコスト増大の課題については，著者らの先行研究 [1] で詳しく論じた．同研究では，ブロックチェーン技術の有効性 [2] や NFT・セキュリティトークンの普及による契約システムの重要性の高まり，IoT・AI 連携による自律的契約管理の可能性について検討し，PoS（Proof of Stake）[3] と DAO（自律分散組織）[4] を組み合わせたトラストレスな契約管理システムを提案する．本研究では，この提案の実用化に向けて，以下の技術的課題に対処した試作システムを開発した．

前回提案からの主要な変更点：

- **コンセンサスアルゴリズムの変更**: PoS の複雑性とカスタマイズの困難さを解決するため，PoA（Proof of Authority）[5] を採用し，独立したアプリケーション層でカスタムステーキングを実装
- **具体的実装環境の構築**: Avalanche Subnet[6] を活用したモジュール型ブロックチェーンの試作
- **省電力性の重視**: Raspberry Pi 環境 [7] での動作により環境負荷を最小化

1.2 研究目的

本研究の目的は，先行研究で提案した契約管理システムを実際に試作し，その技術的実現可能性を実証することである．

具体的には，以下の点を検証した：

- PoA とカスタムステーキングによる「当事者=利用者」エコシステムの実現
- オブジェクト指向型スマートコントラクトによる汎用性の向上
- モジュール型ブロックチェーンによるシステム拡張性の確保

1.3 類似研究

ブロックチェーンを活用した情報管理システムの研究は各方面で進められており，その多くはデータの真正性保証や特定業務の効率化に焦点を当てている．

特許番号 WO2022097608A1 [8] では，ブロックチェーンの分散型台帳上にデータのハッシュ値を記録することで，プライバシーを保護しつつ取引データの真正性を保証する情報管理プラットフォームが提案されている．このアプローチはデータの信頼性確保に有効であるが，システム全体の拡張性や多様な業務ロジックへの柔軟な対応については深く言及されていない．

建設業界における活用事例として，松下ら [9] は ICT 土工を対象に，ブロックチェーンとスマートコントラクトを用いて出来高査定と支払プロセスを自動化する研究を行った．IoT 機器から収集した施工管理情報を基に契約に従った出来高を算出し，ブロックチェーン上に記録することで検査データの信頼性向上を実現している．しかし，この研究は建設業という特定分野に限定されており，他分野への応用には課題が残る．

Muhammad Muneeb らの「SmartCon」[10] は，本研究と同様に第三者機関への依存や不透明性の課題を認識し，スマートコントラクトとトランザクションを管理するトラストレスなフレームワークを提案している．その設計思想は，モジュール型設計やオブジェクト指向の原則を理論的に共有している．

これらの類似研究は，それぞれ特定の課題解決に有効な手法を提案しているが，システム全体の拡張性や多様な契約形態への柔軟な対応，実運用における技術的実現可能性の検証については，さらなる検討が必要とされている．

1.4 基本技術

本研究における主な技術要素であるブロックチェーン，PoS コンセンサス，プライベートチェーン，DAO の基本概念については，先行研究 [1] で詳述した．本稿では，本研究で採用した PoA コンセンサスアルゴリズムとスマートコントラクト，および評価指標として用いるガスコストとガス消費量の定義を述べる．

- **PoA**
  本研究では，コンセンサス層に PoA を採用した．PoA は，事前に承認された信頼できるバリデータがブロック生成を担うコンセンサスアルゴリズムであり，高速なトランザクション処理と低運用コストを可能にする．
- **スマートコントラクト**
  スマートコントラクトは，あらかじめ定められたルールをブロックチェーン上で自動的に実行するプログラムである [11]．本研究では，このスマートコントラクトを用いて，コンセンサス層から独立したアプリケーション層で独自のインセンティブ機構を実装した．
- **ガスコスト，ガス消費量**
  本稿では，スマートコントラクトの実行に必要な計算リソース量を「ガス消費量」として記録・評価した．パブリックチェーンにおいて「ガスコスト」は実質的な経済コストを意味するが，本研究が対象とする PoA プライベートチェーンでは，主にネットワークリソースの乱用を防ぐための内部的な制御指標として扱われる．

2. システム概要

2.1 システム全体像とアーキテクチャ

本研究では，モジュール型ブロックチェーンとオブジェクト指向型スマートコントラクトを中核技術として組み合わせることで，契約プロセスの透明性，信頼性，そして高い適応性を実現した．

システムの基盤には Avalanche Subnet をベースとしたモジュール型ブロックチェーン構成を採用し，契約管理に

特化したプライベートチェーン Gowenet サブネットを構築した．

契約条件はオブジェクト指向型スマートコントラクトとしてブロックチェーン上に登録・実行される．このスマートコントラクトは，サービスとしての適応力，拡張性，および契約ロジックの構造化を実現し，多様な契約形態に柔軟に対応する．契約執行時には，合意に基づきスマートコントラクトが実行され，その結果がブロックチェーンに記録されることで，契約合意の検証と有効性の確認を行う．

2.2 PoA とカスタムステーキングによる「当事者=利用者」エコシステム

本研究では，契約当事者がステークホルダーとしてシステムに参与する「当事者=利用者」エコシステムを提案する．このエコシステムでは，PoA によるシンプルなコンセンサス形成と，アプリケーション層での独自ステーキングを組み合わせることで，参加者自身の貢献によって信頼性を担保する仕組みを構築した．利用者はトークンをステークすることでシステムの安定稼働を支え，その貢献が将来の DAO（自律分散組織）やトークンエコノミーにおける影響力へと繋がるインセンティブ構造を構築した．

2.3 独立したモジュール構成によるセキュアなブロックチェーン

本研究では，ブロックチェーン・ネットワークを独立したひとつのモジュールとして設計し，機能を最小化したうえで，クラウドサービスに依存しない物理的に分離されたブロックチェーンを構築した．これにより，障害発生時の影響範囲を局所化するとともに，モジュール単位での更新や改良が容易になり，全体システムのメンテナンス性およびセキュリティレベルを維持できることを検証した．

2.4 オブジェクト指向型スマートコントラクト

契約管理システムの柔軟性と拡張性を最大化するため，オブジェクト指向型のスマートコントラクト設計を採用した．この手法は，基本的な契約処理のロジックを実装した中央の「ロジックコントラクト」を一度だけデプロイし，個別の契約は軽量な「インスタンスコントラクト」として生成される．各インスタンスは，中央のロジックコントラクトの機能を参照することで動作し，コードの再利用性と保守性を高めることが可能となる．

3. 実証実験の設計と実装

3.1 シングルボードコンピューターによる評価環境の構築

本実証実験では，クラウドサービスに依存しない物理的に独立したブロックチェーン環境を構築するため，シングルボードコンピューター（SBC）である Raspberry Pi を用いた．このアプローチは，専用のインフラを前提とせずに，低消費電力で運用可能な小規模ノード環境を容易に構築できる点に特徴があり，エッジコンピューティング環境におけるブロックチェーンの実現可能性を評価することを目的とする．実験環境として，Raspberry Pi 5 を 3 台使用し，ローカルエリアネットワークを介して接続した．各ノードには Ubuntu OS を導入し，Avalanche ネットワークのコアソフトウェアである AvalancheGo と，その管理ツールである Avalanche-CLI をインストールした．実験環境の主要な仕様を表 1 に示す．

表 1: 実験環境の仕様

項目	仕様
ハードウェア	Raspberry Pi 5 (8GB RAM) × 3 台
OS	Ubuntu 24.04 LTS (64-bit)
ネットワーク	Wi-Fi 6 (IEEE 802.11ax, 5 GHz)
ブロックチェーン	AvalancheGo（v1.11.7，v1.13.2）

本実証実験の複数ホスト環境構築において，AvalancheGo のバージョン互換性に起因する技術的制約に直面した．具体的には，本稿で用いる Avalanche-CLI と AvalancheGo v1.12.x 以降のバージョンの組み合わせでは，独立したローカルネット環境下で C-Chain の同期が完了しない互換性の問題が存在した．これにより，最新の L1 サブネット機能が利用可能な v1.13.2 では，複数ホストによるネットワーク構築が困難であった．一方で，複数ホスト間の同期が安定している v1.11.7 では，Etna アップグレード [12] で導入された L1 サブネット機能に対応していなかった．

この技術的制約を踏まえ，本実証実験では目的別に環境を分離する戦略を取った．Gowenet サブネットの構築とスマートコントラクトの検証は，v1.13.2 を単一ホスト（Pi1）で稼働させる環境で行い，複数ホスト間のピア接続と連携の検証は，安定した v1.11.7 を複数ホスト（Pi2, Pi3）で稼働させる環境で実施した．この戦略により，各バージョンの特性と制約を考慮しつつ，それぞれの検証目的を達成した．

3.2 PoA コンセンサスとカスタムステーキングの実装

本稿で提案する「当事者=利用者」エコシステムを実現するため，コンセンサス形成とインセンティブ設計を実装した．その過程では，当初検討していた PoS に技術的制約が判明したため，アーキテクチャを PoA へ転換するという重要な設計判断を行った．

当初，PoS 環境下で独自の報酬計算ロジック（Reward-Calculator）を実装しようと試みたが，AvalancheGo のコアロジックに依存し、本実証実験のような小規模かつ短期間におけるステーキング値のカスタマイズが困難であった．この課題を回避するため，コンセンサス層には，**PoA** を採用し，Avalanche-CLI ツールを用いて EVM 互換 [11] を有するプライベートサブネット「Gowenet」として実装

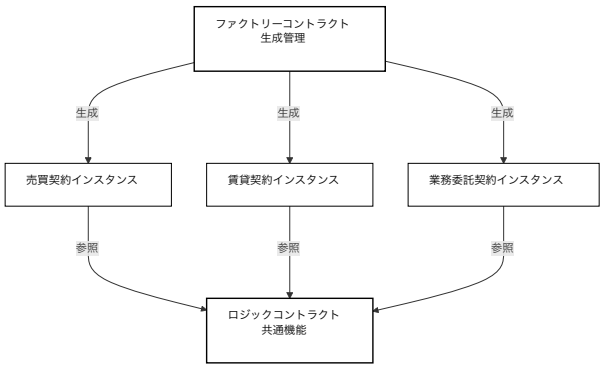


図 1: ロジック分離と参照のアーキテクチャ

した．このサブネットは独自の GOWE トークンを持ち，Avalanche のメインネットワークから独立して動作する．

PoA の採用により，コンセンサス形成のためのステーキングは不要となった．しかし，利用者の参加を促進し，エコシステム内でのインセンティブを設計するため，コンセンサス層から独立したアプリケーション層で独自のステーキング機能をスマートコントラクト（ValidatorIncentives.sol）で実装した．

3.3 オブジェクト指向型スマートコントラクトの実装

本稿で提案するスマートコントラクトは，保守性と拡張性を高めるため，共有ロジックと個別状態を分離するアーキテクチャを採用した．これは，共通の基本機能を定義した BaseAgreement.sol と，具体的な契約種別を実装した Agreement.sol という 2 つの「ロジックコントラクト」を，多数の軽量な「インスタンスコントラクト」が参照して実行する設計である．

このアーキテクチャの中核をなすのが，ファクトリーパターンである．FreelanceContractFactory.sol がその役割を担い，FreelanceContract.sol のインスタンスを動的に生成する．

個別の契約（例：「業務委託契約」）は，それぞれが固有の状態（契約当事者，金額など）のみを保持する，ごく軽量な「インスタンスコントラクト」として生成される．このインスタンスコントラクトはロジックを持たず，全ての処理をロジックコントラクトのアドレスに問い合わせて実行する．この全体像を図 1 に示す．

具体的なプロセスは以下の通りである．そのプロセスを図 2 に示す．

(1) 契約当事者は, FreelanceContractFactory.sol の create-Contract 関数を呼び出す.
(2) 契約個別の情報を引数で渡してコントラクトを生成する.
(3) ファクトリーは，新しい FreelanceContract.sol のインスタンスをデプロイする．このインスタンスは，デプロイ済みの BaseAgreement.sol と Agreement.sol のアドレスをコンストラクタで受け取り，状態変数として保持する.
(4) ファクトリーは，生成したインスタンスのアドレスを呼び出し元に返す.

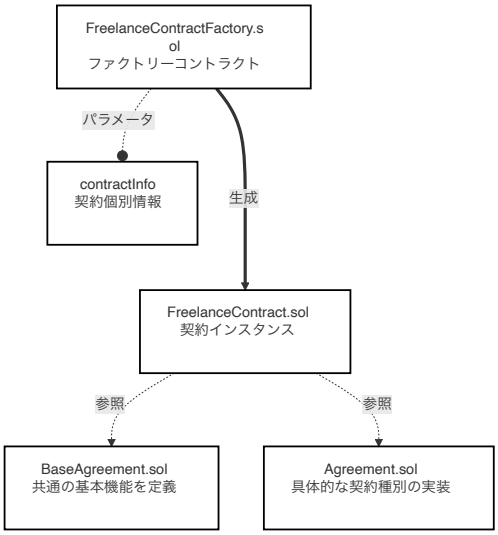


図 2: ファクトリーパターンによるインスタンス生成プロセス

4. 実証実験によって得られる検証結果

4.1 PoA とカスタムステーキングによるエコシステムの検証

本節では，システムの利用者自身が貢献によって信頼性を担保する「当事者=利用者」エコシステムの概念について，その実現に不可欠な 2 つの技術的要件を，PoA とカスタムステーキングの実験を通じて検証した．

4.1.1 PoA による参加者の動的管理

エコシステムの要件である参加者の管理容易性を検証するため，稼働中の Gowenet サブネットに新たなバリデータノードを追加する実験を行った．avalanche blockchain addValidator コマンドで新規バリデータを登録した結果，ネットワークを停止することなく即座に追加され，正常に稼働を開始したことを確認した．これにより，参加者を簡便に管理できるという PoA の運用上の優位性が示された．

実験手順　稼働中のプライマリノード（Pi1）上で, avalanche blockchain addValidator コマンドを用いて新たなバリデータアカウント（gowenet-validator）の P-Chain アドレスと EVM アドレスを関連付けた状態でバリデータを登録した．この際，–remaining-balance-owner パラメータを使用してインセンティブ受取先アドレスを設定した．

実験結果　コマンド実行後，新たなバリデータ（NodeID-7Xhw2mDxu...t3Lg）が即座にネットワークに登録され，Weight: 20，Balance: 1.0 AVAX で動作を開始

```
$ avalanche blockchain addValidator gowenet --local \
  --weight 20 --balance 1.0 --remaining-balance-owner \
  P-custom1pft43ua577v8ql73rz033van8lrr68n0eg260p

ValidationID: 286g5WirtvKaNQdd67oj5LQrKjPDFsJT...HxSV
RegisterL1ValidatorTx ID: 26cf46d65bbuysVHLMzy...6BSb
Waiting for P-Chain to update validator information ...
      100% [===============]

NodeID: NodeID-7Xhw2mDxuDS44j42TCB6U5579esbSt3Lg
Weight: 20
Balance: 1.00 AVAX
Validator successfully added to the L1
```

(a) 新規バリデータ追加実行結果

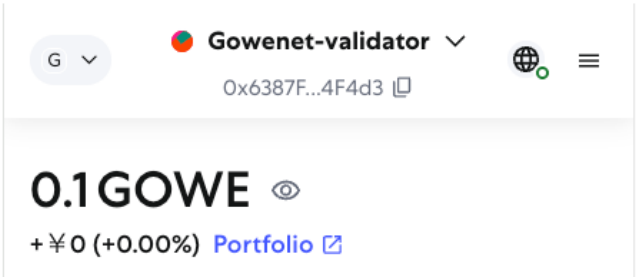
```
$ avalanche validator list gowenet --local

| NODE ID                 | WEIGHT | REMAINING BALANCE |
|-------------------------|--------|-------------------|
| NodeID-2K5fGW177...Dtpb|    100 |         0.099 AVAX|
| NodeID-7Xhw2mDxu...t3Lg|     20 |           1.0 AVAX|
```

(b) バリデータリスト確認結果

図 3: 新規バリデータの追加と確認結果

```
status  0x1 Transaction mined and execution succeed
transaction hash        0x11ca1152a5eca65630...f0b1
block hash      0x30fd6b42a863474ddec1bb5c7b...1fa3
block number    90
from    0x50Bf9Ca2552cECD8BB3265c14c2EB86b31Dc89ad
to      ValidatorIncentives.distributeRewards() 0
    x78e34d6eacd3a8fd99f54ea20b3b21a71785de14
gas     65303 gas
transaction cost        62804 gas
input   0x6f4...a2cd0
decoded input   {}
decoded output   -
logs    []
```

(a) バリデータへの報酬分配のログ



(b) バリデータアカウントの残高を METAMASK で確認

図 4: 新規バリデータの追加と確認結果

した．図 3 に示すように，ValidationID が発行され，P-Chain での RegisterL1ValidatorTx が正常に完了したことを確認した．

4.1.2 アプリケーション層でのインセンティブ設計

次に，参加者がインセンティブ設計に関与できる要件を検証するため，インセンティブ発行コントラクト（ValidatorIncentives.sol）を作成し動作を確認した．オーナーアカウントがバリデータの EVM アドレスを登録後に報酬分配関数を実行した結果，トランザクションは正常に処理され，対象バリデータのアカウント残高が増加したことを確認した．この結果は，コンセンサス層から独立して，アプリケーションの要件に応じた独自のインセンティブ設計を柔軟に実装できることを図 4 が示すように実証している．

実験手順　オーナーアカウント（0x50Bf...）から，ValidatorIncentives.sol コントラクトに対し，バリデータの EVM アドレスを登録する addValidator() 関数を呼び出した．その後，0.1 GOWE トークンを送金額として設定し，登録済みバリデータに報酬を分配する distributeRewards() 関数を実行した．

実験結果　バリデータ登録トランザクションが正常に処理され，続いて報酬分配トランザクションも成功した．実行後，バリデータアカウントの残高が 0.1 GOWE 増加し，アプリケーション層での独立したインセンティブ分配が正常に機能することを確認した．

4.1.3 考察

上記の 2 つの実験結果は，「当事者=利用者」エコシステムの技術的実現可能性を裏付けている．PoA による参加者の動的な管理は，エコシステムへの参加・離脱の自由度を担保する．また，ValidatorIncentives.sol による報酬分配は，コンセンサス層の制約を受けることなく，そのエコシステムに最適化されたインセンティブを設計できる柔軟性を提供する．この 2 つの技術的要素を組み合わせることで，参加者自身の貢献によって信頼性が担保される自律的なコミュニティの基盤を構築できる可能性が示された．

4.2 オブジェクト指向型スマートコントラクト構成による汎用性

本節では，本稿で提案するオブジェクト指向型スマートコントラクトが，従来のモノリシックな実装と比較して，汎用性，保守性，そして経済合理性の観点で優位性を持つことを実験的に検証した．

4.2.1 汎用性の検証：ファクトリーパターンによる動的インスタンス生成

まず，システムの汎用性を検証するため，ファクトリーパターンを用いて新たな契約インスタンスを動的に生成できることを確認した．契約当事者が FreelanceContractFactory の createContract 関数を呼び出した結果，トランザクションは正常に処理され，ContractCreated イベントが発行された．図 5 のイベントログが示す通り，新たに生

```
FreelanceContractFactory.createContract(
    "0x8db97C7cEc...52FC",  // client 契約当事者 (A)
    "0x6A6A35D450...6d62",  // freelancer 契約相手 (B)
    1000000000000000000,    // 1 ETH 契約金額 ()
    "Factory Pattern Test Contract" // 契約内容
)
```

(a) createContract 関数の引数

```
status  0x1 Transaction mined and execution succeed
transaction hash        0xe180a954c4f77f33386d...1649
block hash      0x2eeb71a4d6654e4674099000b28c...5c1b
block number    80
from    0x50Bf9Ca2552cECD8BB3265c14c2EB86b31Dc89ad
to      FreelanceContractFactory.createContract(address
    ,address,uint256,string) 0x228f45d3b8ac77...915b
gas     3176347 gas
transaction cost        3150178 gas
input   0x949...00000
decoded input   {
        "address client": "0x50Bf9Ca2552cECD8B...89ad",
        "address freelancer": "0x096ca79808565...3674",
        "uint256 amount": "1000000000000000000",
        "string description": "TEST"
}
decoded output   -
logs    [
<snipped>
```

(b) ContractCreated イベントログ

図 5: ファクトリーパターンによる契約インスタンス生成実験結果

成された契約インスタンスのユニークなアドレスが記録されており，独立した契約が正しく生成されたことが検証された．これは，単一の窓口から多様な契約要求に柔軟に対応できる高い汎用性を示すものである．

4.2.2 モノリシック型との比較評価

次に，オブジェクト指向設計の保守性と経済性を評価するため，全ロジックを単一のコントラクトに実装する「モノリシック型」と，本研究で採用した「オブジェクト指向型」を比較した．表 2 に全体比較を，表 3 に運用時のガス消費量の詳細を示す．

この評価の妥当性を担保するため，実開発プロセスを模擬し，モノリシック型を基本機能のみのプロトタイプ，オブジェクト指向型を実用機能を網羅したモデルとして設定した．これは，機能追加の過程で生じる保守性の差異を明確にするためである．

表 2 および，表 3 が示す通り，多機能なオブジェクト指向型は初期デプロイで約 6.8 倍，運用時のガス消費量も約 2.4 倍高い．この運用時のコスト増は，主に機能が分離された

表 2: オブジェクト指向型とモノリシック型の比較評価

評価指標	モノリシック型	オブジェクト指向型
総コード行数	113 行（基本機能のみ）	964 行（署名検証・Factory 機能含む）
ファイル数	2 ファイル	5 ファイル
含まれる機能	契約生成・状態管理のみ	署名検証・支払い管理・Factory・動的生成
初回デプロイ時ガス消費量	1,157,704 gas	7,903,923 gas（約 6.8 倍）
修正時ガス消費量	1,157,704 gas（新規デプロイ）	3,039,997 gas（該当モジュールのみ）
データ保持率	0%（アドレス変更）	95%（基盤モジュール再利用）
修正範囲	全体影響確認が必要	対象機能部分
機能の独立性	低（単一ファイルに集中）	高（機能別モジュール分離）
保守性	修正影響範囲が広い	モジュール単位での修正可能
機能追加	全体的な書き換えが必要	該当モジュールのみ修正

表 3: 運用段階での処理ステップごとのガス消費量比較

処理ステップ	モノリシック型	オブジェクト指向型
authenticate	54,419	110,515
deliverWork	36,887	114,855
makePayment	62,012	179,234
completeContract	36,931	58,021
合計	**190,249**	**462,625**

れたモジュール間の外部コール（external call）に伴うオーバーヘッドに起因する．しかし，この初期コストの優位性は，システムの修正・拡張フェーズにおいて逆転する．このことは，開発中に発生した実際のバグ修正事例で明確になった．貢献度スコア計算の不具合に対し，オブジェクト指向型では影響範囲が限定され，問題の特定から修正を完了するまでに要した時間は約 30 分であった．表 2 の通り，同様の修正をモノリシック型で行った場合に必要と推定されるガス消費量と比較して多いが，データ損失もなく，外部システムへの影響も回避できた．

4.2.3 考察

上記の実験結果は，オブジェクト指向設計が初期のガス消費量増加というトレードオフを補って余りある，長期的な優位性を持つことを実証している．この初期コスト増が実用上の問題とならない理由は，本研究が採用する PoA プ

ライベートチェーンの特性にある．

本稿で扱う「ガスコスト」は，パブリックチェーンにおける実質的な経済コストとは本質的に異なる．パブリックチェーンでは，ガスコストは不特定多数のマイナーへのインセンティブとして機能するが，参加者が限定された本システムのプライベートチェーンでは，ネットワークリソースを統制するための内部的な制御指標である．計算資源を消費しない PoA ではバリデータが高いガスコストをインセンティブとする必要もなく，ガスの価格は運営主体によって極めて低く設定可能である．

したがって，本研究の環境下ではガスコストの絶対額は軽微である．そのわずかな増加と引き換えに得られる保守性，拡張性，そして総保有コストの大幅な削減という長期的価値は，極めて大きい．機能がモジュールとして独立しているため，機能追加や修正時の影響範囲を局所化でき，開発・保守効率が大幅に向上する．この保守性の優位性は，システムの継続的な運用において累積的な効果をもたらすものである．

4.3 モジュール型ブロックチェーンによる拡張性

本節では，Avalanche Subnet を用いたモジュール型ブロックチェーンが，システムの独立性と拡張性の観点で優位性を持つことを実証した．特に，実環境で直面したバージョン互換性の課題を背景に，本アーキテクチャの有効性を論じる．

4.3.1 システム拡張性の検証

本稿で提案するモジュール型アーキテクチャの拡張性を検証するため，単一モジュールにおけるシステム拡張性の実験を行った．前章で述べた通り，複数ホスト間の同期が安定している v1.11.7 環境を選択し，稼働中のプライマリホストに対し，セカンダリホストをバリデータとして追加登録した．

その結果，ネットワークを停止することなく，新規バリデータとして正常に追加され，ブロック生成に参加し始めたことを図 6 に示すように確認した．

実験手順 稼働中のプライマリホスト（Pi2）に対し，avalanche blockchain addValidator コマンドを用いてセカンダリホスト（Pi3）のノード ID をバリデータとして登録した．

実験結果 コマンド実行後，Pi3 は即座に Pi2 とのピア接続を確立し，ブロックチェーンの同期を開始した．図 6 に示すログの通り，ネットワークを停止させることなく，Pi3 が新たなバリデータとして正常に追加され，ブロック生成に参加し始めたことを確認した．

4.3.2 考察

上記の実験結果は，本稿で提案するモジュール型ブロックチェーンアーキテクチャが，高い拡張性を持つことを実証している．稼働中のネットワークを停止させることなく，動的にバリデータを追加できたことは，将来の負荷増大や可用性向上の要求に対し，個別のモジュール単位で柔軟にスケールアウトできる能力を示すものである．

```
Pi2 (Primary):   192.168.3.86  NodeID-7Xhw2mD...t3Lg
Pi3 (Secondary): 192.168.3.75  AvalancheGo v1.11.7
```

(a) 拡張性の実験環境

```
# ホストでの起動 Pi3AvalancheGo
Pi3$ avalanchego --network-id=1337 \
--bootstrap-ips=192.168.3.86:9651 \
--bootstrap-ids=NodeID-7Xhw2mD...t3Lg
```

(b) バリデータ追加プロセス

```
Pi2: {"result": {"isBootstrapped": true}}
Pi3: {"result": {"isBootstrapped": true}}

Network Status: Healthy, Nodes: 2
```

(c) 同期確認結果

図 6: 拡張性の実験における環境・プロセス・確認結果
注：省略記号 ‘...‘は NodeID の中間部分を省略したことを示す．

このアーキテクチャの優位性は，開発過程で直面したバージョン互換性の課題への対処においても示された．従来のモノリシックなシステムでは，一部の機能が特定のバージョンに依存する場合，システム全体の機能制約に直結する．しかし本稿のモジュール型アーキテクチャでは，機能ごとに最適なソフトウェア（バージョン）で独立したブロックチェーンを構築するというアプローチが可能である．今回，拡張性の検証（v1.11.7）とサブネット機能の検証（v1.13.2）を分離して実施できたこと自体が，技術的課題に対する耐障害性の高さを実践的に証明している．

これらの結果から，本稿のモジュール型ブロックチェーンは，単一モジュールの拡張性と，モジュール間の独立性による耐障害性を両立しており，持続的に成長可能なシステムを構築する上で極めて有効なアーキテクチャであると結論付けられる．

5. 総括と貢献

本稿では，従来の契約管理システムが抱える課題を解決するため，モジュール型ブロックチェーンを用いた新しい契約管理システムを提案し，その設計と試作を行った．提案システムは，Avalanche Subnet を基盤とし，SBC 上で稼働するプライベートチェーンに，高速な PoA コンセンサス，アプリケーション層でのカスタムインセンティブ設計，そしてオブジェクト指向型スマートコントラクトを組み合わせたものである．

実証実験を通じて，提案システムの基本アーキテクチャが実現可能であることを示すとともに，PoA とカスタムインセンティブによるエコシステムの実現可能性，オブジェクト指向設計による保守性と経済合理性，そしてモジュール型アーキテクチャの拡張性と耐障害性を明らかにした．

本研究の貢献は，実用的な分散型契約管理システムを構築するための，具体的かつ総合的な設計指針を実証に基づき提示した点にある．本稿で提案したアーキテクチャは，以下の点で従来技術に対する優位性を持つ．

- **セキュリティと拡張性の両立**: 独立したサブネットによるモジュール構成は，外部の脆弱性から隔離された高いセキュリティを実現すると同時に，必要に応じてノードやサブネットを追加し，システム全体を停止させることなくスケールアウトできる拡張性を提供する．プライベートチェーンの採用も，企業間の機密情報を含む契約データを安全に取り扱う上で不可欠である．
- **環境配慮と低コスト運用**: 計算集約的な PoW を避け，SBC と PoA を組み合わせることで，システムの消費電力を大幅に抑制した．これは，持続可能性が求められる現代において重要な特性であり，エッジデバイスでの自律的なシステム運用という将来的な応用への道筋を示すものでもある．
- **高い汎用性と保守性**: オブジェクト指向設計をスマートコントラクトに導入することで，コードの再利用性を高め，多様な契約要件に迅速かつ低コストで対応できる汎用的な基盤を実現した．これにより，一度構築したシステムを長期間にわたって運用・発展させていく上で，極めて重要な保守性を確保した．

これらの優位性により，本研究は，セキュリティ，拡張性，コスト，そして環境負荷という多角的な要請に応える次世代の分散型契約管理システムの実現に向けた，重要な一歩となるものである．

今後の展望

今後は，IoT・AI 連携による契約履行の高度化や，サプライチェーン等の他分野への応用を進めるとともに，スマートコントラクトの法的有効性や知的財産戦略といった社会的・制度的課題への対応も重要な検討事項となる．これらを通じて，ブロックチェーン技術の社会実装に向けた新たな道筋を提示できると考える．

参考文献

[1] 藤本浩司, 井上敦司, 三淵啓自: “ブロックチェーンを用いた契約管理システムの開発,” 情報処理学会研究報告, Vol.2025-DPS-202, No.16, Vol.2025-CSEC-108, No.16, pp.1–8, 2025 年 3 月.

[2] World Economic Forum, “The Future of Financial Infrastructure: An Ambitious Look at How Blockchain Can Reshape Financial Services,” 2016. Available: 入手先 ⟨http://www3.weforum.org/docs/WEF_The_future_of_financial_infrastructure.pdf⟩.

[3] Kiayias, A., Russell, A., David, B., & Oliynykov, R. (2017). Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In Advances in Cryptology – CRYPTO 2017 (LNCS Vol. 10301, pp. 357–388). Springer.

[4] Jentzsch, C. (2016). The DAO. Retrieved from 入手先 ⟨https://www.dao.org/whitepaper.pdf⟩

[5] Wood, G.: Kovan PoA Consensus Engine. GitHub Wiki. 入手先 ⟨https://github.com/kovan-testnet/kovan-scenarios/wiki/Kovan-PoA-Consensus-Engine⟩

[6] Ava Labs: Avalanche: A Novel Metastable Consensus Protocol Family for Cryptocurrencies, Whitepaper, 2018. 入手先 ⟨https://www.avalabs.org/whitepaper⟩ オンライン版.

[7] Raspberry Pi Foundation, Raspberry Pi 5 (8GB), Raspberry Pi, 2023. [Online]. Available: 入手先 ⟨https://www.raspberrypi.com/products/raspberry-pi-5/⟩.

[8] 情報管理プラットフォームシステム及び，その処理方法, WO 2022/097608 A1, 国際特許公報, 2022 年. 入手先 ⟨https://patents.google.com/patent/WO2022097608A1/ja⟩ オンライン版.

[9] 松下 文哉, 小澤 一雅, “ICT 土工を対象とするブロックチェーンとスマートコントラクトを用いた出来高査定及び支払プロセス自動化のためのシステム開発,” 土木学会論文集 *F4*（建設マネジメント）, 78 巻, 1 号, pp.133–147, 2022 年, 公開日: 2022/07/20, Online ISSN: 2185-6605. DOI: 入手先 ⟨https://doi.org/10.2208/jscejcm.78.1_133⟩, URL: 入手先 ⟨https://www.jstage.jst.go.jp/article/jscejcm/78/1/78_133/_article/-char/ja⟩.

[10] M. Muneeb, Z. Raza, I. Ul Haq, and O. Shafiq, ”SmartCon: A Blockchain-Based Framework for Smart Contracts and Transaction Management,” IEEE Access, vol. 10, pp. 23687-23699, Mar. 2022.

[11] Buterin, V.: A Next Generation Smart Contract and Decentralized Application Platform (2013).

[12] Ava Labs: Avalanche: Avacloud. Etna Upgrade - Avalanche API Documentation. [Online]. 入手先 ⟨https://developers.avacloud.io/data-api/etna⟩. Accessed on August 20, 2025.