情報処理学会論文誌　デジタルプラクティス　Vol.3　No.2　1–11 (Apr. 2022)

ブロックチェーンを活用した企業間ワークフロー管理システムの実行エンジン

長野 裕史[1]　下沢 拓[2]　島村 敦司[2]　薦田 憲久[3]

受付日 2021年8月24日, 再受付日 2021年10月28日,
採録日 2021年12月14日

概要：今日，業務効率の向上を目的とした企業内ワークフローのシステム化が進展している．一方，企業間ワークフローは，依然として紙の書類に基づいて手作業で処理されている．これは，単一組織が運用するシステム上のプロセスとデータを他の組織が信頼できないためである．本稿では，ブロックチェーン上のスマートコントラクトを用いた，企業間ワークフロー実行エンジンの実装方式を提案する．提案方式では，ワークフロー処理の全ステップにおいて，関連するデータの一貫性を逐次保証することにより，ワークフローのライフサイクル全体にわたってデータの一貫性を保証する．提案方式を実装し，処理性能とデータサイズを評価した結果，企業間ワークフロー処理において実用可能であることを確認した．

キーワード：ブロックチェーン，ワークフロー，システムセキュリティ，システムアーキテクチャ

Execution Engine for Cross Organizational Workflow Management System on Blockchain

Hirofumi Nagano[1]　Taku Shimosawa[2]　Atsushi Shimamura[2]　Norihisa Komoda[3]

Received: August 24, 2021, Revised: October 28, 2021,
Accepted: December 14, 2021

Abstract: Currently, workflows in enterprise business processes have become to be implemented as information systems to improve the efficiency of processing. However, workflows between organizations are still processed manually based on paper documents because it is difficult for the organizations to trust the process and the data in the system owned by one of the organizations. In this paper, an implementation method of workflow execution engine for cross organizational workflow is proposed using smart contract on blockchain. It secures the data consistency among organizations throughout the workflow lifecycle by verifying the consistency of all the related data among nodes in every step of workflow. Through the implementation of proposed method and the evaluation of the data size and the performance, it is confirmed that the proposed method is applicable for practical use in cross organizational workflow.

Keywords: blockchain, workflow, system security, system architecture

1. はじめに

情報技術の進展に伴い，企業内のビジネスプロセス，特にワークフロー処理のシステム化が進展している．ワークフローとは，ある提案案件に対する起案，承認，差し戻し，却下などの処理を，あらかじめ定義された処理フローに基づいて処理する業務プロセスである．ワークフロー管理は，ビジネスプロセス管理の主要機能の１つであり，様々な業務分野に適用されている [1], [2]．ワークフロー管理をシステム化することにより，処理の迅速化と工数の削減が可能となる．しかし，企業間でのワークフローは，依然として紙の書類を用いて人手で処理されている．これは，ワークフロー管理システムを含む Business Process Management System（BPMS）が集中型のシステムで構築され，企業内利用に限定されているためである [3]．たと

[1] Hitachi America, Ltd.
Research & Development Division, Hitachi America, Ltd., California, United States
[2] 株式会社　日立製作所　研究開発グループ
Research & Development Group, Hitachi Ltd., Kokubunji, Tokyo 185–8601, Japan
[3] コーデソリューション株式会社
Code Solutions Co., Ltd., Osaka 550–0002, Japan

えば，企業間ワークフローの一例である受発注業務は，見積書や注文書，請求書などの書類を，紙や電子メールで送受信することで処理されている．第三者による電子商取引サービスも提供されているが，この場合は，サービス提供事業者がワークフロー管理システムを集中管理することになるため，データの改ざんの可能性を排除できない．別の例として，顧客に対して複数の企業が連携してサービスを行うようなケースにおいて，顧客情報の登録や案件処理などを企業間で行う場合がある．このようなワークフローも，現在，企業間で個別に書類や電子メールを用いることで処理されている．これは，特定の企業や第三者がワークフローシステムを管理した場合，管理企業によるデータの改ざんの可能性を排除できないためである．

これに対し，商用ワークフロー管理システムを連動させることで，企業間のワークフローシステムを実現する取り組みが行われており [4]，データフォーマットの標準化などが進められている．しかし，この場合，システム間で送受信されるデータ以外は，各システムで個別にデータを保有しているため，他社システムでのデータの改ざんを検知することはできない．

暗号資産の流通基盤として開発されたブロックチェーン技術 [5] は，複数組織間で改ざん不可能なデータを共有できるという特徴を持つ．この特徴を利用し，企業間ワークフローをブロックチェーン上に実装する取り組みが行われている．文献 [6] には，企業間ワークフローの一例である貿易取引を，ブロックチェーン上に実装する方式が提案されているが，複数企業間でワークフロー管理を実行するためのシステムアーキテクチャは示されていない．これに対し，文献 [7] には，BPMS とブロックチェーンを連携するアーキテクチャにより，企業間でのワークフロー処理における処理結果の透明性と耐改ざん性を保証する方式が提案されている．しかし，BPMS は前記のとおり集中型のシステムで構築されているため，ワークフロー定義は特定企業の管理に依存する．このため，改ざんのない正しいワークフロー定義に基づいて処理されたことを保証できない．一方，文献 [8] では，ワークフロー実行エンジンにブロックチェーン上のスマートコントラクトを用い，処理結果だけでなく処理プロセスも非中央集権化することで，ワークフロー処理の信頼性を担保する方式が提案されている．しかし，本方式が対象とするのは，ワークフロー定義を企業間で合意し，その合意に基づいてワークフローを実行し，その処理結果を監査するという，ワークフローの一連のプロセスのうちの実行フェーズのみである．ワークフローの処理結果を全参加企業が信頼可能とするためには，定義から実行，監査に至るワークフローのライフサイクル全体にわたって，特定のユーザや企業に依存する Single Point of Trust（SPoT）を排除する必要がある．

上記の課題に対し，筆者らは，ワークフローの定義，実行，監査処理をそれぞれブロックチェーン上のスマートコントラクトとして実装し，これらを連動することにより，企業間ワークフロー管理における SPoT を排除し，データの改ざんを防止するシステムアーキテクチャを提案し，攻撃シナリオに基づくセキュリティ評価を行っている [9]．提案アーキテクチャでは，ワークフロー実行処理の各ステップにおいて，ノード間でデータを照合しながら処理する．また，ワークフローの定義自体もワークフローとして定義することにより，定義の起案，承認フローを通じてスマートコントラクトを生成し，各ノードに配布する．また，監査処理についても，他のブロックチェーンノードとデータを照合して取得する．これらの処理方式により，ワークフローのライフサイクル全体にわたって，参加企業間で信頼できる処理プロセスとデータを共有することが可能となる．しかし，スマートコントラクトの処理フローやデータ構造などの具体的な実装方式や性能は明確でない．

本稿では，上記アーキテクチャで動作するワークフロー実行スマートコントラクトの実装方式として，情報の信頼性を考慮した方式を提案し，実用性を評価する．

以下，2 章では，ブロックチェーンを活用した企業間ワークフロー管理システムにおけるスマートコントラクトの実装上の課題を整理し，3 章でこれを解決する逐次データ保証型ワークフロー実行エンジンの実装方式を提案する．その後，4 章で実装評価の結果を述べる．

2. 企業間ワークフロー管理システムの実装課題

2.1 企業間ワークフロー管理システムのアーキテクチャ

ワークフロー管理システムは，定義（Definition），実行（Execution），監査（Auditing）にわたるワークフローのライフサイクル全体を管理するシステムであり，一般に**図 1** に示すように，ワークフロー定義（Workflow Definition）を作成するワークフロー設計ツール（Workflow Design Tool），定義に基づきワークフロー実行を制御するワークフロー実行エンジン（Workflow Execution Engine），処理結果を保管するデータベース，およびクライ

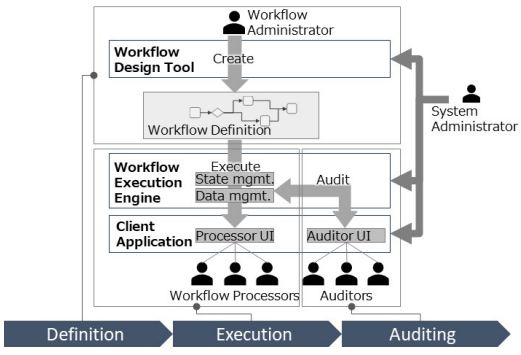


図 1　ワークフロー管理システムの概要
Fig. 1　Overview of workflow management system.

アントアプリケーション（Client Application）から構成される [2].

企業間ワークフローは，同一企業内のワークフロー処理者によって処理される企業内プロセスと，企業内での処理結果に基づいて企業間をまたがって処理される企業間プロセスから構成される．筆者らは，ブロックチェーンを用いることで，処理実行における特定の企業による SPoT を排除しつつ，処理結果の改ざんを防止する企業間ワークフローを実現する方法として，**図 2** に示すシステムアーキテクチャを提案している [9].

本アーキテクチャでは，各企業（Org.A，Org.B，Org.C）は，図 2 の上部にある内部ワークフローシステム（Internal WF System）と，図 2 の下部にあるブロックチェーンを用いた企業間ワークフロー処理システム（Cross Organizational WF System）のブロックチェーンノード（Blockchain node）の 2 つのシステムを保有し，内部ワークフローシステムでの処理結果をもとに，企業間ワークフロー処理システムを用いて企業間で処理を実行することにより，全体のワークフロー処理を実現する．

各企業は，企業内ワークフローで承認された結果をもとに，企業間ワークフローを実行するブロックチェーンノード上の実行エンジン（Execution Engine）にリクエストを送信する．実行エンジンは，各参加企業のノード上でスマートコントラクト（Smart Contract）として動作し，ワークフロー定義（WF Definition）に基づいて処理を実行し，結果を各企業のノード間で照合したうえで，各ブロックチェーンノード上のデータベース（State DB）とブロックチェーン（blockchain）に記録する．このアーキテクチャにより，企業間ワークフロー管理システムにおける SPoT を排除し，処理プロセス，データの両方の改ざんを防止することで，参加企業が信頼できる企業間ワークフローシステムを実現できる．

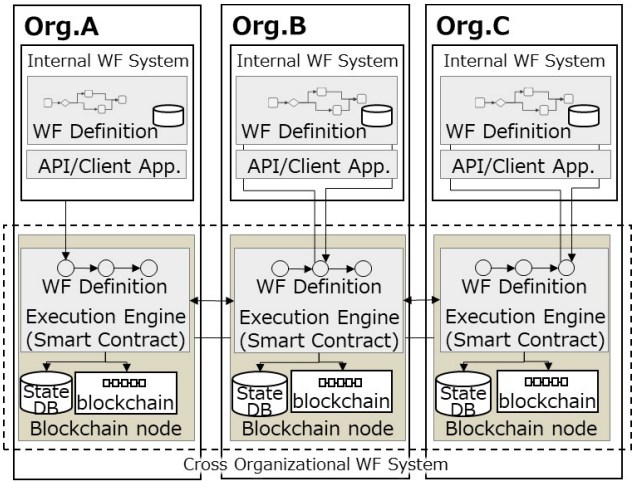


図 2　企業間ワークフロー管理システムのアーキテクチャ

Fig. 2　Cross organizational workflow management system architecture.

2.2　ワークフロー実行処理の概要と課題

次に，図 2 に示すアーキテクチャにおいて，ワークフローを実行処理する際の処理フローを説明する．本アーキテクチャでは，あらかじめ企業間で合意し，各ノードに配置された定義に基づいて，ワークフロー実行スマートコントラクトがワークフロー処理を実行する．**図 3** にワークフロー実行処理の概要を示す．なお，図中の丸数字付きの矢印は，企業 A（Org.A）のワークフロー処理者がリクエストを送信する場合の処理の流れのみを示しており，図を見やすくするため，すべてのリンクは記載されていない．

まず①において，ワークフロー処理者は，承認・却下などのワークフロー処理リクエストを Web アプリケーション（WebApp）経由でワークフロー実行スマートコントラクト（SC）に送信する．ワークフロー実行スマートコントラクトは，②において，処理リクエストを他の企業 B（Org.B）と企業 C（Org.C）のブロックチェーンノード（Blockchain node）上のワークフロー実行スマートコントラクトに配信する．各ノードのワークフロー実行スマートコントラクトは，当該トランザクションを実行する前のワークフロー処理ステータスなど，更新前のステート DB（State DB）参照データと，実行後の処理ステータスや次の処理ステップなど，ステート DB の更新データをセットにした Read/Write セットを作成し，送信元である企業 A のノードにこれを返信する．ここで作成される Read/Write セットの例を**図 4** に示す．

この例では，Action1 という処理レコードについて，ステータス（Status）が要求済み（Requested）である更新前のレコードと，承認後にステータスが承認済み（Approved）となり，起案内容（Contents）が修正された更新後のレコードの両方をまとめたデータが Read/Write セットとなる．

図 3 に戻り，一定数の返信を受信した送信元ノードのワークフロー実行スマートコントラクトは，各ノードからの Read/Write セットを比較し，それらが一致する場合

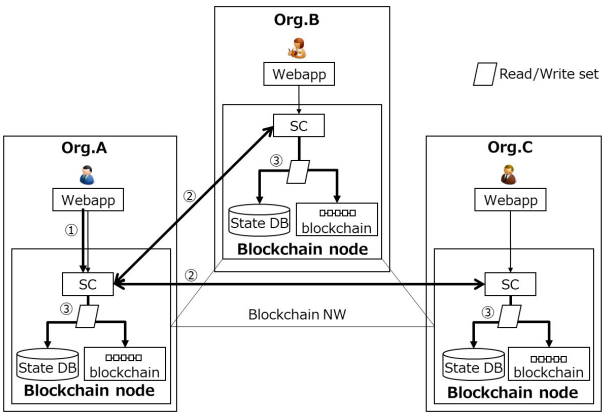


図 3　ワークフロー実行処理フロー

Fig. 3　Processing flow of workflow execution.

に，③において Read/Write セットを各ノードに配信し，各ノードにてブロックチェーンファイルに追加するとともに，ステート DB を更新する．ステート DB は最新のレコードを格納したデータベースである．一方，ブロックチェーンファイルは，トランザクションデータをブロックとしてまとめ，ハッシュ値でつないだデータ構造であり，改ざんが困難な履歴ファイルとしてステート DB の復元や監査等に用いることができる．

特定企業のシステム管理者によって，ワークフロー実行スマートコントラクト，あるいは処理ステータスなどの処理結果が改ざんされていた場合，他のノードと Read/Write セットが一致しないため，処理が完了しない．これにより，ワークフロー定義フェーズで合意したプロセスどおりにワークフロー処理が実行され，その処理結果が各ノード間で一致していることを，すべての参加企業が信頼することができる．

しかし，各トランザクションにおいてノード間で照合する Read/Write セットは，図 4 に示したとおり，当該トランザクションにおいて更新するレコードの範囲に限定される．たとえば，ある起案に対する承認トランザクションを実行する場合，ワークフロー承認者は起案データを参照したうえで承認処理を行うが，ブロックチェーン上の更新レコードが承認アクションのステータスを記録するレコードのみの場合，ノード間で照合される Read/Write セットには参照した起案データは含まれない．このため，起案データが特定のノードで改ざんされており，承認者が改ざんされたデータを参照して承認したとしても，参照データについてはノード間で照合されないため，他のノードは改ざんされたデータに基づく承認であることに気づけない．このように，ワークフロー処理の信頼性を担保するために，ノード間で照合する必要のあるデータの範囲を適切に定義し，これを Read/Write セットとして生成し，照合するためのデータ構造，およびスマートコントラクトの処理設計が課題となる．

以下，3 章では上記の課題を解決するワークフロー実行エンジンの実装方式について述べる．

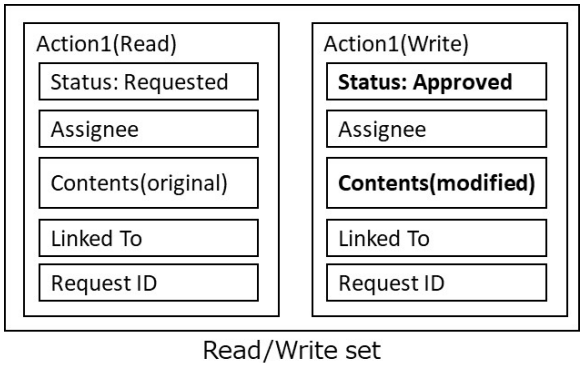


図 4　Read/Write セットの例
Fig. 4　Example of Read/Write set.

3. 逐次データ保証型ワークフロー実行エンジン

3.1 逐次データ保証型ワークフロー実行エンジンの考え方

トランザクション処理時にノード間で照合する Read/Write セットは，当該トランザクション実行時に更新するステート DB のレコードの更新前後のデータである．ステート DB のデータ構造としては，起案，承認，却下などの処理ごとにレコードを分け，これらのレコードをポインタでつなぐ**図 5** に示す構成案 (a) と，ワークフロー案件に対する処理すべてを 1 レコード内にまとめる**図 6** に示す構成案 (b) が考えられる．

構成案 (a) では，ワークフロー案件（Case）の起案（Request）と，その後の承認，却下などの処理（Action）のたびに 1 レコードが生成される．Request レコードには，当該案件が起案中か，完了済みかなどのステータス（Status），起案内容（Contents），および後続のレコードへのリンクを示す Linked To の情報が格納される．Action レコードには，案件の ID（Request ID）が付与されるとともに，レコード間は後続のレコードを示す Linked To の情報で関連付けられる．また，当該処理が処理待ちか処理済みかを示すステータス（Status），処理者（Assignee），および起案内容（Contents）が格納される．起案内容については，Request レコードに提案時の内容が記録され，そ

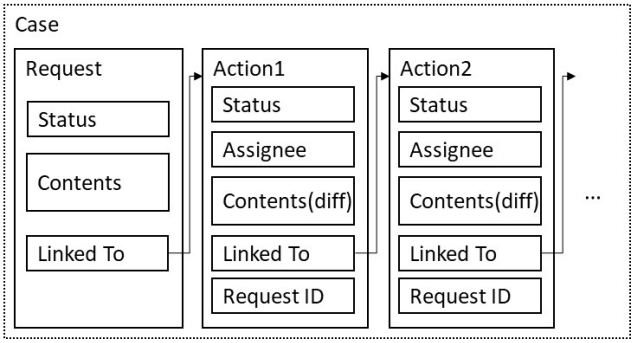


図 5　ステート DB のデータ構造案 (a)
Fig. 5　Data structure of State DB (a).

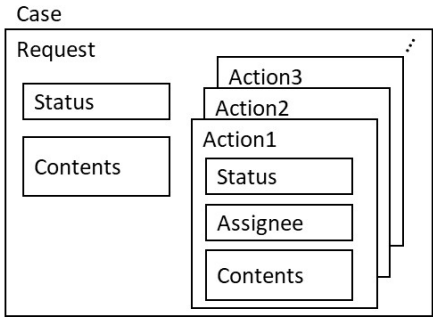


図 6　ステート DB のデータ構造案 (b)
Fig. 6　Data structure of State DB (b).

の後の Action では，更新がある場合にのみ，その差分のみが記録される．これにより，最小限のデータ量でトランザクションを処理していくことが可能となる．

一方，構成案 (b) では，各案件（Case）のデータは 1 つの Request レコード内にすべて格納される．ワークフローの起案時に，起案内容である Contents と Status を格納した Request レコードが生成されると，その後の Action のたびに，この Request レコード内に Action データが追記されていく．各 Action には，構成案 (a) と同様，Status と Assignee が格納されるとともに，当該 Action の処理時点での Contents が逐次格納される．これにより，対象ワークフロー案件に関するすべてのデータを，Request ID を指定することで一括取得することができる．それぞれのデータ構造案における Read/Write セットの作成対象範囲を**図 7**，および**図 8** に示す．

構成案 (a) の場合，トランザクション処理時に生成される Read/Write セットは，図 7 の点線に示すとおり，当該処理である Action1 の更新前後のレコードと，生成される次の Action2 から構成される．当該ワークフロー案件に関するレコードのうち，上記以外のレコードについては，Read/Write セットに含まれず，ノード間で照合処理が行われない．このため，たとえばある企業が承認リクエストを処理する際に，この企業のノード上で，起案時の Request ID のコンテンツがシステム管理者によって改ざんされていた場合，改ざんされた起案内容に基づいて承認処理がなされ，承認後にシステム管理者がこれを隠ぺいするためにコンテンツを元に戻すというような不正の可能性がある．このように，構成案 (a) では，ワークフロー実行中のデータの信頼性に関して課題がある．

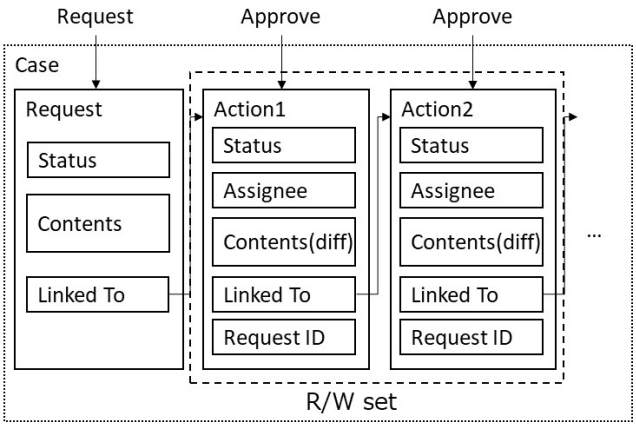


図 7　データ構造案 (a) における Read/Write セット
Fig. 7　Read/Write set in data structure of State DB (a).

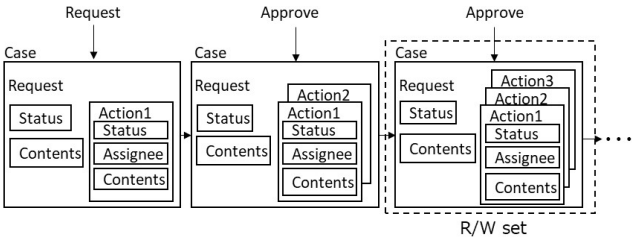


図 8　データ構造案 (b) における Read/Write セット
Fig. 8　Read/Write set in data structure of State DB (b).

一方，構成案 (b) の場合，トランザクション処理時に生成される Read/Write セットは，図 8 に示すとおり，当該時点での最新の Request レコードの更新前後のデータとなる．Request レコード内には，当該ワークフロー案件のそれまでの処理履歴がすべて含まれている．このため，ワークフローの各ステップを処理するたびに，当該ワークフロー案件の過去の処理履歴も含めて，すべてのデータをノード間で照合しながら処理することになる．これにより，ワークフローの各処理者が Request や Action の処理をする際に，そのステップでの処理内容だけでなく，どのような Contents に対して処理を行ったのか，またどのような処理履歴を参照したうえで処理を行ったかも含めて，すべての参加企業間で逐次データの整合性を保証しながら処理をすることが可能となる．しかし，ワークフローのステップが進むたびに，Request レコードのサイズが増大し，それに伴って Read/Write セットのサイズも増大する．このため，処理性能への影響やストレージ容量の増大が懸念事項となる．これについては 4 章で評価する．

次節では，構成案 (b) に基づいた逐次データ保証型ワークフロー実行エンジンの処理の詳細について述べる．

3.2　逐次データ保証型ワークフロー実行エンジンの詳細

図 9 に，構成案 (b) を用いた逐次データ保証型ワークフロー実行エンジンの処理フローを示す．ここでは，図 9 の左上にあるワークフロー案件の Case a について，Action-1 として定義された承認依頼を割り当てられたユーザが，ワークフロー実行エンジンを通じて承認処理を行うケースを例に，構成案 (b) を用いることでどのようにデータの整合性が保証されるかを示す．

まず，ワークフロー実行エンジンは，処理 (1) においてユーザ権限を確認したのち，処理 (2) において，その時点で当該ユーザに割り当てられているワークフロー案件の一覧を取得する．Action データの処理者（Assignee）に当該ユーザが指定されており，かつ当該 Action の Status が起案中（Open）の Action を含む Request レコードを指定して取得することで，当該ユーザに割り当てられている処理待ちの全案件のデータを取得することができる．ユーザは処理対象案件を選択し，承認／却下の処理内容を Status に指定し，必要に応じて Contents を更新した Action レコードを，処理 (3) においてワークフロー実行エンジンに送信する．

これを受信したワークフロー実行エンジンは，処理 (4) において，ステート DB に格納した当該案件のワークフロー定義に基づいて，送信された Action レコードのコン

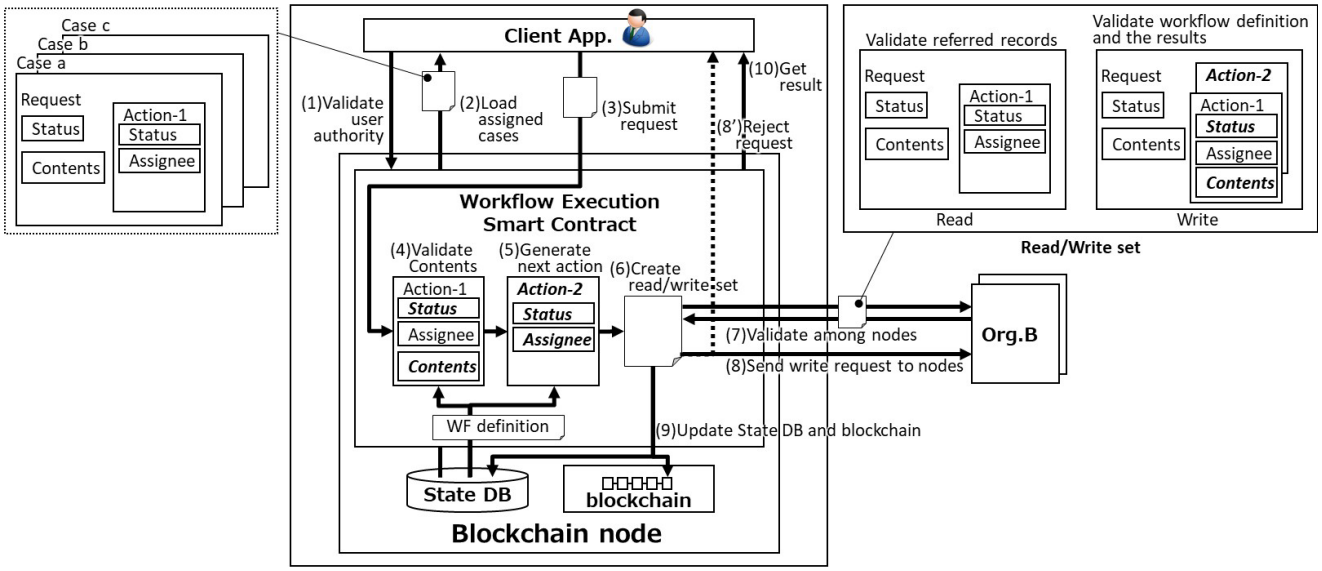


図 9　ワークフロー実行スマートコントラクトの処理フロー

Fig. 9　Processing flow of workflow execution smart contract.

テンツを検証する．問題がない場合は，処理 (5) において，ワークフロー定義に基づいて次の処理を Action として生成し，Assignee などの情報をセットした上で，処理 (6) においてこれらの情報をまとめて Read/Write セットを生成する．図 9 では，このユーザが，当該ワークフロー案件の起案に対する次の処理（Action-1）を処理し，次の承認者の処理（Action-2）を生成する例を示している．

次に，処理 (7) において，他の参加企業のノードとの間で Read/Write セットの照合を行う．Read/Write セットのうち，Read 部分は，Request レコードの更新前のデータであり，この案件の過去の処理履歴として，元々の起案データと，ユーザが処理する前の Action のデータが含まれる．これを他のノードと照合することにより，当該ユーザが Action-1 を処理するにあたり参照した当該案件の起案コンテンツや過去の処理履歴が，ノード間で一致していることを保証できる．一方，Read/Write セットのうち，Write 部分には，当該ユーザによる処理内容である Action-1 の更新部分と，次の処理として生成された Action-2 のデータが含まれる．これを他のノードと照合することにより，Action-1 の処理内容と生成された Action-2 の情報から，各ノードのワークフロー定義にも改ざんがないことを確認できる．

処理 (7) の照合処理の結果，他ノードと Read/Write セットが一致しない場合，処理 (8') においてユーザに Reject を送信する．システム管理者は，不一致部分をもとに，どの部分に改ざんがあったかを確認することができる．処理 (7) の照合処理が一致した場合には，処理 (8) において Write データを他ノードに配信するとともに，処理 (9) において，自ノードのステート DB と blockchain を更新し，処理 (10) において結果をユーザに送信する．

処理 (9) において，特定のノードでステート DB，または Blockchain への書き込みに失敗し，整合性が取れなくなった場合，他のノードとの間で Read/Write セットの不一致が発生する．しかし，この場合でも，合意形成に十分なノード数が正常に動作していれば，トランザクションは処理は正常に継続される．不整合が発生したステート DB，または Blockchain については，他のノードのデータを用いて復旧することができる．

このように，提案方式では，ワークフローのステップごとに，その時点における当該案件のデータ全体，およびワークフロー定義に基づく次のステップの Action を含む更新データをノード間で逐次確認する．これにより，処理結果とプロセスの両方のデータの信頼性を，ワークフローの実行時に逐次保証するワークフロー実行エンジンを実現できる．

4. 逐次データ保証型ワークフロー実行エンジンの評価

本章では，データ改ざん耐性の評価の後，提案方式によるワークフロー実行エンジンを用いて，具体的な企業間ワークフローの例である複数企業間での顧客情報登録ワークフローを実装し，これを用いてデータ量と処理時間を評価し，提案方式の実用性を検証する．

4.1 データ改ざん耐性の評価

提案方式により，特定の企業に依存する SPoT が排除され，データの改ざんが防止されることを，定義フェーズ，実行フェーズ，監査フェーズからなるワークフローのライフサイクルに沿って評価する．

文献 [9] で示したとおり，提案方式によるアーキテク

チャにおいて，攻撃対象となる資産やデータは，図 2 に示す構成要素のうち，内部 WF 管理システム，企業間 WF 定義，ステート DB，Blockchain の 4 点である．このうち，内部 WF 管理システムは，各企業の管理資産であり，内部監査で適切に管理すべき対象である．このため，その他の 3 つの資産について，ワークフローの各フェーズにおける SPoT の有無と改ざん耐性の評価を行う．

まず，定義フェーズでは，企業間 WF 定義が各企業のノードに配信される．配信された WF 定義を各企業のシステム管理者等が改ざんした場合，実行フェーズでのトランザクション実行時に Read/Write セットの結果がノード間で不整合となり，改ざんを検知できる．たとえば，特定企業の管理者が，自ノード上の WF 定義において，他の企業の承認ステップをスキップするような改ざんを行った場合，図 6 に示すデータ構造において，当該ステップの Action の Assignee が他のノードとは異なるデータとなるため，Read/Write セットの不整合として検知できる．このように，定義フェーズでの SPoT は排除され，改ざんリスクがないことが分かる．

次に，実行フェーズにおいては，特定の企業のシステム管理者が，たとえば他の企業による承認ステップをスキップする目的でワークフロー定義を改ざんしたり，自ノード上のステート DB に格納された起案データや処理結果，Blockchain 上の処理履歴を改ざんするケースが考えられる．まず，ワークフロー定義を改ざんした場合は，定義フェーズにおけるケースと同様，トランザクション実行時のノード間での不整合により検知できる．次に，ステート DB や Blockchain 上のデータを改ざんした場合，3.2 に示したとおり，提案方式の構成案 (b) では，Read/Write セットに当該ワークフロー案件のすべてのデータが含まれるため，過去の処理履歴も含めてノード間での照合時に検知することができる．たとえば，特定企業のノード上で，過去の承認時の Contents が改ざんされていた場合，その後の処理ステップを実行する際に，図 9 に示す処理フローの処理 (7) において，他のノードで生成された Read/Write セットとの間で，Read データが不整合となるため，改ざんを検知することができる．このように，実行フェーズにおいても SPoT を排除し，改ざんを防止することができる．

監査フェーズにおいては，完了済みの処理履歴が，ステート DB と Blockchain 上で改ざんなく保管されていることが重要となる．提案方式では，監査フェーズの参照トランザクションにおいても，実行時と同様，ワークフロー案件単位ですべてのデータを取得し，これをノード間で照合するため，特定企業のノードでデータが改ざんされていた場合に検知することができる．このため，監査フェーズでも SPoT は排除されており，データの改ざんが防止される．

上記のとおり，提案方式によれば，ワークフローのライフサイクル全体にわたって，特定の企業に依存する SPoT を排除し，データの改ざんを防止することが可能である．

4.2　評価に用いる企業間ワークフローのユースケース

今回のデータ量，および処理時間評価に用いた企業間ワークフローのユースケースを**図 10** に示す．このワークフローは，顧客に対して 3 社の企業が連携してサービスを行うようなケースにおいて，最初にサービス企業間で顧客情報の登録・承認を行う業務処理であり，従来は書類や電子メールを用いて企業間で共有・確認し，各企業で個別にシステム登録していた処理である．本ユースケースでは，まず Step1 において，参加企業のうちの一社である企業 A（Company A）から顧客登録リクエスト（Customer Registration Request）が送信されると，Step2 において，企業 B（Company B）がこの内容を確認したうえで承認（Approve Registration），あるいは却下（Reject Registration）を行う．企業 B で承認された場合は，同様にして企業 C（Company C）が確認を行い，承認／却下の処理を行う．企業 C が承認し，最終的に 3 つの企業全ての承認が得られると，顧客として登録され，ワークフローが完了する．

4.3　データサイズの評価

3.1 節で述べたとおり，ステート DB のデータ構造には，図 5 に示した構成案 (a) と，図 6 に示した構成案 (b) がある．構成案 (a) は，ワークフローの各ステップにおいて生成される差分データのみを Read/Write セットとして生成するため，ステート DB，および blockchain のデータ量を最小限にできる．一方，提案方式で用いる構成案 (b) によれば，ワークフロー実行時のデータの信頼性は担保されるが，データ量が増大する問題がある．本節では，図 10 に示した顧客情報登録のワークフローを実装する際に，シス

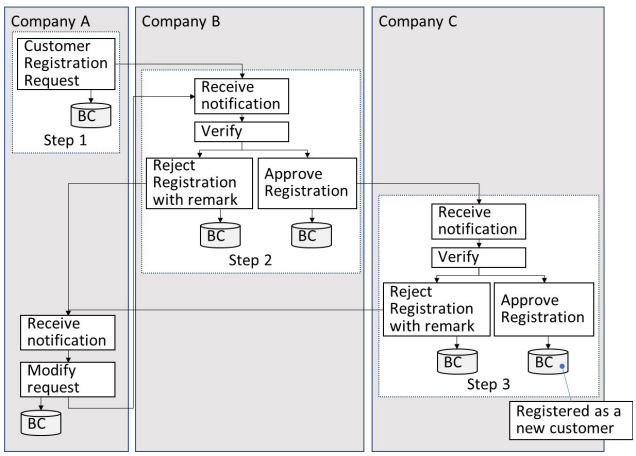


図 10　企業間ワークフローのユースケース
Fig. 10　Use case of cross organizational workflow.

表 1　レコードサイズ（bytes）
Table 1　Record size (bytes).

	Contents	Header	Total
Request	1024	32	1056
Action	1024	48	1072

テム全体で必要とされるデータ容量を，構成案 (a) と構成案 (b) それぞれの場合について評価する．

このワークフローの処理で使用される各データのサイズを，**表 1** のように設定する．Contents は，本ワークフローで登録する顧客情報，およびワークフロー処理時に追加される処理者のコメントなどのデータである．顧客氏名，住所，電話番号，などの基本属性に 200 Bytes，その他の属性は 1 項目あたり 50 Bytes を 10 項目，コメントに 300 Bytes 程度を確保できるデータサイズであり，添付ファイルを用いる場合は，そのリンクとハッシュ値の情報をその他の属性として格納することもできる．このため，一般的な顧客情報登録に十分なサイズであると言える．Header は，それぞれ Request，Action レコードの Contents 以外のデータであり，Status や LinkTo の情報の他，更新日時や登録者などの情報を含む．構成案 (b) については，Request レコードに Action レコードが含まれるが，以下では Request レコードは Action データ以外の部分を表すこととする．

- R：Request レコード全体のサイズ（bytes）
- A：Action レコード全体のサイズ（bytes）
- A_H：Action レコードのヘッダサイズ（bytes）
- A_C：Action レコードのコンテンツサイズ（bytes）

とすると，構成案 (a) におけるワークフローのステップ n での Read/Write セットのサイズ RW_{an} は，まずステップ 1 の Request トランザクションで，

$$RW_{a1} = R + A_H \tag{1}$$

となる．ステップ 2 以降は，更新前後の Action レコードのサイズと次のステップの Action レコードの合計となる．更新後の Action レコードのサイズは，各ステップにおけるコンテンツ更新の有無によって変動するが，承認のみでコンテンツの更新を含まない場合，ヘッダ内の Type を承認済みに変更するのみであり，Read/Write セットのサイズは，

$$RW_{an} = A_H \times 3 \tag{2}$$

で一定となる．

一方，構成案 (b) の Read/Write セットのサイズ RW_{bn} は，初回の Request トランザクションは構成案 (a) の場合と同様，

$$RW_{b1} = R + A_H \tag{3}$$

次回以降は，

$$RW_{bn} = (R + A \times (n-2) + A_H) + (R + A \times (n-1) + A_H) \tag{4}$$

となる．

次に，ステート DB のサイズについては，構成案 (a) の場合，ステップごとに Action レコードが 1 件ずつ追加されていくため，ステップ n におけるステート DB のサイズ S_{an} は，

$$S_{an} = RW_{a1} + (A_H \times (n-1)) \tag{5}$$

となる．一方，構成案 (b) では，Contents を含む Action レコードが Request レコード内に追加・更新されていくため，ステップ n におけるステート DB のサイズ S_{bn} は，

$$S_{bn} = RW_{b1} + (A \times (n-1)) \tag{6}$$

となり，各トランザクションにおいて，コンテンツサイズ分，構成案 (a) よりサイズが大きくなる．

最後に，Blockchain のデータサイズは，トランザクションごとに Read/Write セットが追加されていくため，R/W セットの累積のデータ量となり，構成案 (a) の場合のステップ n における Blockchain データサイズ B_{an}，構成案 (b) の場合のステップ n における Blockchain データサイズ B_{bn} は，それぞれ，

$$B_{an} = \sum_{I=1}^{n} RW_{ai} \tag{7}$$

$$B_{bn} = \sum_{I=1}^{n} RW_{bi} \tag{8}$$

となる．

これらの数式と，表 1 に示した顧客情報登録のユースケースにおけるレコードサイズに基づいて，ワークフローの各ステップにおけるそれぞれのデータサイズを計算した結果を**表 2** に示す．ワークフローのステップ数に応じたデータ量の増大を確認するため，図 10 で示した 3 ステップ（Action-2 まで）のワークフローにおけるデータ量に加え，ワークフローのステップ数が 10 の場合の 10 ステップ目（Action-9）のデータ量を計算した．

構成案 (a) のデータ構造では，R/W セットは対象ステップの更新前後のデータのみのため，ワークフローのステップによらず一定である．このため，R/W セットの累積となる Blockchain データのサイズも線形で増加するのみであり，ステート DB と Blockchain を合わせた Storage 容量は 10 ステップ目でも約 4 kB/件である．月 1000 件のワークフローを処理した場合でも，4 kB × 1000 × 12 = 48 MB/年である．

なお，従来型の集中型のワークフロー管理システムで同様のレコードサイズ，ステップ数のワークフローを処理した場合，処理結果を格納するデータ量はステート DB と同様であり，また処理履歴ファイルに蓄積されるデータ量も構成案 (a) の場合の blockchain に記録されるデータ量とほぼ同等であると考えられる．ただし，ブロックチェーンを使う構成案 (a) の場合では同じデータ量がすべてのノード

表 2　データサイズ（bytes）
Table 2　Data size (bytes).

	R/W set		ステート DB		blockchain		Storage	
	(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)
Request (Step 1)	1104	1104	1104	1104	1104	1104	2208	2208
Action-1 (Step 2)	144	3280	1152	2176	1248	4384	2400	6560
Action-2 (Step 3)	144	5424	1200	3248	1392	9808	2592	13056
Action-9 (Step 10)	144	20432	1536	10752	2400	107808	3936	118560

で蓄積されるのと比べ，システム全体で蓄積されるデータ量はこの 1/3 になる．つまり，3 ノードからなるこの例では，48×3＝144 MB/年となる．

一方，構成案 (b) では，各ステップで処理されるコンテンツも含めた Action レコードが，Request レコード内に追加・更新されていくため，この更新前後のデータを合わせた R/W セットのサイズがステップごとに増加していく．さらに，R/W セットの累積である Blockchain データもこの累積で増加する．Blockchain サイズは 3 ステップ目で 9,908 bytes，Blockchain とステート DB を合わせた全体の Storage 容量は，13,056 bytes/件となり，10 ステップ目で約 119 kB/件となる．しかし，月 1000 件のワークフローを処理した場合でも，119 kB × 1000 × 12 = 1,428 MB/年となる．3 ノードからなるシステム全体のデータ量でもその 3 倍の 4,284 MB/年であり，企業システムのデータ容量としては問題ないと言える．ただし，Contents のデータ容量によってはこれがさらに増大する可能性はある．しかし，ワークフローに添付される PDF ファイルや画像ファイルのような大容量データは，外部ストレージに格納し，ブロックチェーンシステムにはこのファイルへのリンクと改ざん防止のためのハッシュ値のみを格納する方法が一般的である．このような方法を採用することにより，ブロックチェーンシステムとしての Storage 容量を抑制することができる．あるいは，大容量データをクライアントアプリケーション側で圧縮・解凍処理することで，ブロックチェーンシステムへの格納データ容量を削減することができる．ただし，この場合はクライアントアプリケーション側で圧縮・解凍の処理時間が発生することになる．

4.4　性能評価

ワークフロー実行，および監査の処理性能を評価するため，提案方式によるワークフロー実行エンジンを，コンソーシアム型ブロックチェーン基盤である Hyperledger Fabric [10] を用いて実装し，書き込みデータサイズと処理性能の関係を評価した．実測環境の構成を**表 3** に示す．合意形成方式（Endorsement Policy）には，過半数の承認（Endorsement）をもって合意とみなす方式を採用した．

表 3　実測環境構成
Table 3　Experiment environment.

Item		Spec
Hardware	CPU	Xeon 8175M (2.5GHz) x 2 cores
	Memory	8GB
Software	OS	Ubuntu 18.04.5 LTS
	Blockchain	Hyperledger Fabric 2.2.1
	Database	CouchDB 3.1.0
	Language	Node.js (client/chaincode)
Blockchain Network Configuration		3 Organizations on a single instance
Endorsement policy		2 out of 3

これを実現するための最小ノード数は 3 であり，3 ノード中 2 ノードの承認をもって合意とみなすことができる．この 3 ノードを，クラウド上の 1 インスタンス上に実装したブロックチェーンネットワークを構成し，評価に用いた．

この実測環境を用いて，図 9 の処理フローにおいて，処理 (3) でクライアントアプリケーションがリクエストを送信してから，処理 (10) で結果を受信するまでの応答時間を，ステート DB への書き込みデータ量を変動させて計測した．実際の運用では，各企業がブロックチェーンノードを保有するため，ノードはネットワーク上に分散して配置される．このため，処理フローのうち，(3) のリクエストの送信，(7) における処理結果の収集，(8) の書き込みリクエストの配信，および (10) の結果の受信はネットワーク遅延の影響を受けるが，今回の構成では，この点は考慮していない．上記の測定条件による性能計測の結果を**図 11** に示す．

提案方式における書き込みデータは，当該ステップにおいて更新される Request レコードである．構成案 (b) では，ステート DB 上で 1 件の Request データが更新されていくため，このデータサイズは，ステート DB のサイズと同一であり，表 1 に示すとおり，3 ステップ目で，3,248 bytes，10 ステップ目で 10,752 bytes である．図 11 に示すとおり，書き込みサイズが 10 kB までは，応答時間はほぼ 100 ms で変動がなく，書き込みサイズが 1 MB の

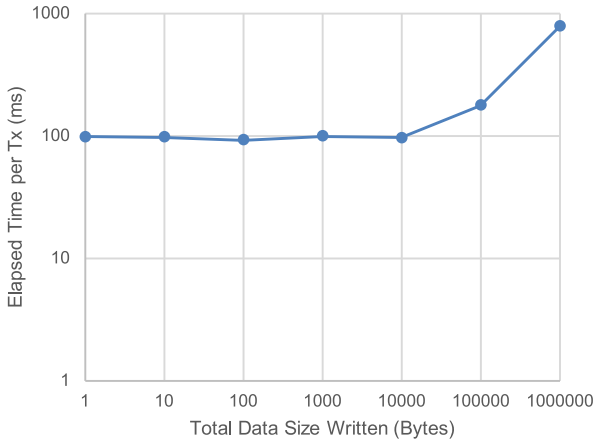


図 11　データサイズごとの処理時間
Fig. 11　Invoke time per data size.

場合には，700 ms 程度に増加する．書き込みデータサイズが影響する処理は，図 9 の処理フローにおいて，主に (9) のステート DB，および Blockchain へのデータ書き込み処理である．書き込みデータサイズが 10 kB までは，(9) の処理時間が他の処理時間よりも相対的に小さく顕在化せず，10 kB を超えると，(9) の処理時間がトランザクション処理の主要な要素として顕在化していると考えられ，10 kB 以降はデータサイズに比例して応答時間が増加すると考えられる．10 ステップ目の書き込みデータサイズが 1 MB になるリクエストデータサイズは，式 (6) によると，約 100 kB である．4.3 節で述べたとおり，PDF ファイルなどの大容量データは外部ストレージへ格納し，ハッシュ値だけをリクエストデータに含めることにより，データサイズを 100 kB 程度に抑え，ブロックチェーンシステムへの書き込みデータ量を 1 MB 程度に抑えることは可能である．書き込みデータ量を削減する別の手段として，大容量データを圧縮したうえでリクエストデータに含めることも可能だが，この場合はクライアントアプリケーション側で圧縮・解凍のための処理時間が追加で発生する．

今回の測定に含まれていないネットワーク遅延の影響は，一般に国内の通信において RTT（Round Trip Time）が 100 ms 未満であることを考えると，図 9 の処理フローにおける (3)，(10) のクライアントとの間の通信，および (7)(8) のノード間の通信のそれぞれで 100 ms 程度として 200 ms 程度の増加にとどまる．これを，データサイズが 100 kB の場合の計測結果に加えても 900 ms であり，実環境におけるネットワーク遅延を考慮しても提案方式による処理性能は実用上問題ないと言える．

また，企業数に応じてノード数が増加する場合があるが，(3) のリクエスト送信，およびその後の各ノード上での処理は各ノードで並行して実行されるため，ノード数に応じた処理時間の増加は発生しない．また，今回の評価で採用したような過半数による合意形成アルゴリズムを用いた場合，過半数の結果の一致を合意とみなすため，応答時間のばらつきによる影響を低減することができる．

一方，従来型の集中型のワークフロー管理システムで同様のワークフローを処理した場合の処理時間は，主に各トランザクションで更新されるデータベースの処理時間であると想定される．この際に処理されるデータサイズは，ステップ数によらず，表 1 の Request，あるいは Action レコードのサイズである 1 kB 程度となる．これは，ブロックチェーンを活用したシステムでデータ書き込み処理時間が顕在する 10 kB 以下である．このため，集中型のワークフロー管理システムでは，ネットワーク遅延を除けばステップ数によらず 100 ms 以下で処理できると想定される．しかし，システムを保有する企業によるデータ改ざんのリスクが排除できない．提案方式では，データ改ざんのリスクを排除したうえで，リクエストデータサイズが 100 kB 程度でも，700 ms／件という実用上問題ない性能を実現できることが分かった．

次に，監査時の処理性能について評価する．監査時には，各ワークフロー案件について，すべての処理データを取得する必要がある．構成案 (b) のデータ構造によれば，各ワークフロー案件のデータは Request レコードにすべて格納されているため，1 トランザクションで 1 件のデータを取得できるため，ワークフロー実行時のトランザクションと同様の処理時間で監査向けデータの取得を行える．

提案方式では，月 1000 件のワークフロー案件を処理した場合，年間 12,000 件の取得に要する時間は，100 ms * 12,000 = 1,200 sec（20 min）となる．従来，企業間ワークフローに対する監査業務では，対象企業でのデータ取得に加え，相手先企業のデータも個別に取得し，これらを突き合わせる作業が必要だった．しかし，提案方式によれば，企業間で一致が確認されたデータを 20 分で取得可能であり，監査業務を大幅に効率化できると考えられる．

5.　実践で得られた知見

本稿では，複数のトランザクションから構成される企業間ワークフロー処理において，企業間のデータの整合性を業務プロセス全体で保証する方式として，ワークフロー単位で構成したレコードを逐次照合する処理方式を提案した．

提案方式によれば，定義，実行，監査からなるワークフローのライフサイクル全体にわたって，特定の参加企業によるデータの改ざんを防止できることが分かった．また，このような処理方式とした場合でも，書き込みデータサイズが 10 kB 程度までであれば，ネットワーク遅延を考慮しても処理性能への影響はないことが分かった．このデータサイズは，1 kB 程度のレコードを処理するワークフローで 10 社目の承認を行う際の書き込みデータサイズに相当する．従って，参加企業数 10 社の企業間ワークフローの

運用に性能上問題がないこと意味する．参加企業数が 10 社を超えるようなワークフローは，書類の差し戻し等を考慮すると非効率となるため，最大でも 10 社程度のワークフローに分割することが想定される．このため，本稿で想定した書き込みデータサイズ 10 kB 程度で対応可能であり，実用上性能面での問題がないことが分かった．

6. まとめ

本稿では，筆者らが提案するブロックチェーンを活用した企業間ワークフローシステムのアーキテクチャで動作するワークフロー実行スマートコントラクトの実装方式として，逐次データ保証型ワークフロー実行エンジンを提案した．提案方式を用いることで，ワークフローの実行中に，すべての関連データが企業間で整合性が取れた状態で処理されていることを保証することができる．提案方式では，ワークフローのステップ数が増加した際の保存データ量の増大と処理性能の低下が懸案となるが，複数企業間での顧客情報登録ユースケースを想定した実装評価の結果，保存データ量，処理時間ともに実用上問題ないことを確認した．

参考文献

[1] 電気学会ワークフロー調査専門委員会（編）:「ワークフローの実際」，日科技連出版 (1999).

[2] 速水治夫，渋谷亮一，鈴木登雄，生駒順一，寺下陽介，植野直樹，金子　聡，林　　潔：ここまで来たワークフロー管理システム（3）ワークフロー製品の実際，情報処理，Vol.40, No.5, pp.507–513 (1999).

[3] Stohr, A. E. and Zhao, J. L.: Workflow Automation: Overview and Research Issues, *Information Systems Frontiers*, Vol.3, No.3, pp.281–296 (2001).

[4] 速水治夫，勝間田仁，世古将洋，提箸公代，渋谷亮一，石丸知之，大南正人，岡田謙一：商用ワークフロー管理システムと連動するインターワークフロー支援システム，情報処理学会論文誌，Vol.41, No.10, pp.2708–2718 (2000).

[5] Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System, available from 〈https://bitcoin.org/bitcoin.pdf〉 (2008).

[6] Fridgen, G., Radszuwill, S., Urbach, N. and Utz, L.: Cross-Organizational Workflow Management Using Blockchain Technology - Towards Applicability, Auditability, and Automation, *Proc. the 51st Hawaii International Conference on System Sciences* (2018).

[7] Alves, P., Paskin, R., Frajho, I., Miranda, Y., Jardim, J., Cardoso, J., Tress, E., Ferreira da Cunha, R., Nasser, R. and Robichez, G.: Exploring Blockchain Technology to Improve Multi-party Relationship in Business Process Management Systems, *Proc. the 22nd International Conference on Enterprise Information Systems* (*ICEIS 2020*), Vol.2, pp.817–825 (2020).

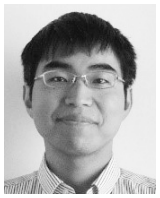
[8] Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A. and Mendling, J.: Untrusted Business Process Monitoring and Execution Using Blockchain, *International Conference on Business Process Management*, pp329–347 (2016).

[9] Nagano, H., Shimosawa, T., Shimamura, A. and Komoda, N.: Reliable Architecture of Cross Organizational Work-Flow Management System on Blockchain, *IADIS International Journal on Computer Science and Information Systems*, Vol.15, No.2, pp.29–43 (2021).

[10] The Linux Foundation: Hyperledger Fabric - Hyperledger, available from 〈https://www.hyperledger.org/use/fabric〉 (2017).

長野 裕史（非会員）

1998 年東京大学大学院総合文化研究科広域科学専攻修士課程修了．同年株式会社日立製作所入社．2019 年より，Hitachi America, Ltd., Research & Development Division Director．現在，金融機関向け情報システムの研究開発に従事．



下沢 拓（正会員）

2012 年，東京大学大学院情報理工学系研究科コンピュータ科学専攻博士課程修了（博士（情報理工学））．同年株式会社日立製作所入社．現在，エンタープライズシステム向けプラットフォーム技術の研究開発に従事．

島村 敦司（非会員）

1999 年東京工業大学大学院情報理工学研究科計算工学専攻修士課程修了．同年株式会社日立製作所入社．現在，企業向け情報システムの研究開発に従事．

薦田 憲久（非会員）

1974 年大阪大学大学院工学研究科電気工学専攻修士課程修了．同年株式会社日立製作所入社．1991 年大阪大学助教授，1992 年 8 月同大学工学部教授．2002 年，同大学大学院情報科学研究科マルチメディア工学専攻教授．情報システムの研究に従事．2015 年，大阪大学名誉教授．コーデソリューション株式会社顧問．工博．電気学会 2000 年度進歩賞などを受賞．IEEE，電気学会の終身会員．