# Intercompany Business Transaction Platform Architecture: Concept and Design

Attila Franko, Adam Mate, Pal Varga
Department of Telecommunications and Artificial Intelligence,
Budapest University of Technology and Economics,
Műegyetem rkp. 3., H-1111, Budapest, Hungary

***Abstract*—Handling processes for distributed, heterogeneous endpoints with certain centralized decision-making possibilities is a challenge for enterprise services. Many companies already use digital contracting, automated quotation, and purchase order (PO) systems – although following the process and being aware of the status is still troublesome for both the end-users and the company controllers as well.**

**This paper describes a B2B Transaction Platform Architecture concept and realization, fitting to current high-scale enterprise requirements. The foundational elements of this framework consist of atomic events and transactions that combine to form comprehensive business processes. Key elements include activity-based process modeling, immutable node structures, publish/subscribe notification systems, and aggregation mechanisms for unambiguous process-state tracking. Balancing distributed execution with centralized decision-making, this system ensures transactional integrity between stakeholders such as purchasers, suppliers, and financial operators. This architecture aims to handle the competing demands of multiparty transaction coordination, scalability, and centralized administrative control while leveraging the benefits of a distributed cloud infrastructure.**

***Index Terms*—intercompany transactions, application management, enterprise multiparty service handling**

## I. Introduction

As many consultancies [1] and market demand confirm, the world is open to the development of automated accounting systems and data-driven automation, so that human resources can be used for more value-added work, rather than for administration. Therefore, many are thinking about platform economy and management solutions that can be used even on a global scale (much research has been published on their feasibility [2], [3]). This paper is a slice of this digitization of a transaction management solution.

The BUTLER (Intercompany BUsiness Transaction pLatform architEctuRe) framework, developed through collaboration between SAP and the Budapest University of Technology and Economics (BME), represents a distributed, cloud-based architecture designed to manage multi-party business transactions. Its core objectives include resolving concurrency challenges, ensuring unambiguous process-state visibility across stakeholders, and providing centralized control over complex transaction workflows. These objectives are key to successful *enterprise-scale* transaction handling. The platform leverages atomic activities, hierarchical branch structures, and a publish-subscribe notification system to handle scenarios such as purchase order (PO) workflows. BUTLER's design emphasizes scalability, interoperability with legacy systems, and configurable aggregation logic to compute global process states.

## II. Related works

Transaction-oriented approaches have been a cornerstone in business and financial systems for many years. Research literature shows how companies have responded to business needs by breaking down transaction layers into smaller units, often called micro-transactions [4]. These micro-transactions represent various business activities that occur within systems, such as processing payments, managing contracts, tracking status information, and handling status changes.

Within the blockchain technology space, researchers have developed innovative systems to enhance financial processes. One notable example is the Blockchain Financial Statements (BFS) framework [5], which was designed to improve financial transparency, reduce potential fraud, and automate the verification of transactions. This represents an important step forward in applying blockchain to financial contexts.

The existing literature highlights several important limitations. Current blockchain technologies often struggle with speed and performance issues when handling large volumes of transactions [6]. This creates a significant barrier to their adoption in enterprise environments where transaction throughput is critical [7]. Business transaction protocols such as RosettaNet PIP and ebXML CPA exist in the literature, these primarily focus on ensuring transaction network success and determining the correct ordering of transactions rather than addressing the comprehensive needs of financial systems.

The research landscape thus reveals notable gaps: while blockchain offers theoretical advantages for financial transaction systems through its inherent security and immutability features, practical implementations continue to face performance challenges that limit widespread adoption. Current solutions have not adequately bridged this divide between blockchain's security benefits and the performance requirements needed for enterprise-level financial applications. Additionally, the microtransaction-oriented environment can be further developed and adapted to players performing a wide variety of business functions. Blockchains have been widely suggested to be used in supply chains – including healthcare [8], logistics [9] and various other perspecitves [10] –, however proof of concepts with enterprise systems integration that include legacy scenarios, are missing.

978-3-903176-75-1 ©2025 IFIP

## III. Requirements for Transaction Handling

### A. General Requriements

The platform must resolve concurrent microtransactions between stakeholders while maintaining a unified process state visible to all authorized parties. Core requirements include atomic event handling, centralized transaction ordering, and integration of the legacy system via adapter nodes. The architecture prohibits community segregation, allows dynamic partner relationships, and mandates human-interpretable aggregation logic for process statuses.

The Purchase Order (PO) process serves as the primary use case, encompassing preprocesses from Request for Quotation (RFQ) to payment closure. Partial deliveries and staggered payments require hierarchical transaction structures, where subsets of goods and payments triggers status change.

The transactions are used as atomic events involving all the necessary data to conclude a contract.

## IV. Architectural Components & Considerations

### A. Core elements

*1) Central server nodes:* SAP-managed cloud resources enforce transaction ordering and process synchronization. These utilize distributed databases with eventual consistency models to handle replicated workloads.

*2) Client nodes:* Stakeholder endpoints (e.g., supplier ERP systems, LSPs) initiate transactions and consume process-state updates. Role-based access controls restrict visibility per transaction hierarchies.

*3) Message broker infrastructure:* Publish/subscribe nodes distribute state changes via topics aligned with business processes. QoS guarantees include exact-once delivery and offline client synchronization.

### B. Integration challenges

The seamless coordination of logistical activities among supply chain partners and SAP's internal systems is paramount for operational efficiency. A foundation of trust, satisfaction, and commitment among stakeholders is essential to foster effective logistics integration. The BUTLER structure emerges as a key enabler in this context, offering a configurable and adaptable framework that facilitates smooth integration and synergy between SAP systems (and other ERPs).

This approach improves the overall cohesion of supply chain operations, allowing for the following. The below-summarised points support the overall integrability of the systems.

- Improved data flow and communication - Service layers modernization,
- Enhanced visibility across the supply chain - Leafs,
- Streamlined processes and reduced inefficiencies - Data access layers (DALs),
- Greater adaptability to changing business needs (APIs, more easy-to-reach needed data).

Using the flexible infrastructure of the BUTLER structure, organizations can achieve a higher degree of integration, ultimately leading to more responsive and efficient supply chain management.

### C. Integration with SAP systems

The importance of synchronized logistical activities among supply chain members and SAP internal systems is crucial. Trust, satisfaction, and commitment are vital for supporting effective logistics integration. BUTLER structure offers seamless integration between SAP internal systems synergy thanks to its configurable and flexible infrastructure.

### D. Integration with legacy systems

Legacy systems using EDI (Electronic Data Interchange) or XML protocols require protocol translation to BUTLER's JSON-based activities, introducing latency and computational overhead.

Possible solutions for protocol translation:

- XML-to-JSON Conversion
- Data Model Discrepancies - resolving field-matching.

### E. Partner system integration

As with legacy systems, standard, documented, predefined forms of communication can be used.

### F. Stakeholder Management

Actors are categorized into:

- Owners: Create/Modify root transactions (i.e., superuser),
- Observers: Read-only access to designated branches,
- Actors: Child transaction initiators under parent nodes.

Role conflicts are mitigated through immutable node policies. Once a transaction spawns children, parent modifications are prohibited, except via new branch creation.

## V. Transaction ordering challenges

Central servers resolve timing conflicts through:

- Lamport timestamps: logical clocks; causal ordering [11]
- Retransmission queues: detecting gaps via periodic client-server synchronization,
- Idempotent operations: ensuring duplicate transactions yield identical states.

## VI. Concept and Design of BUTLER Architecture

### A. Activity-Centric Design

The BUTLER architecture employs an activity-centric design to model intercompany business transactions, where every process, message, or action is represented as an atomic "activity." These activities form hierarchical structures (branches) that enable granular tracking of multistep workflows while ensuring centralized governance. This approach resolves concurrency challenges in distributed environments by combining immutable transaction logging with configurable aggregation rules, achieving a significant reduction in reconciliation errors compared to traditional EDI (Electronic Data Interchange) systems.

Activities are defined using JSON schemas that contain mandatory and optional fields, enabling flexibility across use cases like Purchase Orders (PO) or Advanced Shipping Notifications (ASN). Each activity includes:

- Identification: A 128-bit UUIDv4 for global traceability.,
- Temporal Markers: Hybrid timestamps combining Lamport counters and NTP-synced UTC values,
- Main data of PO : statuses and prices, estimated delivery,
- Role-Based Access Controls: observers (read-only access) and actors (child transaction initiators),
- Critical Field Markers: Attributes like activity, status triggering real-time alerts via the Publish/Subscribe Notification System (PSNS),
- Child Transaction Registry: Lists of spawned subprocesses.

Each activity is a JSON object with mandatory fields (uuid, status, parties) and optional metadata. This allows parallel processing across server nodes without cross-activity locks.

```
{
  "uuid": "47512891-83ef-4eae-b67b-507f379c8064",
  "activity": {
    "name": "Purchase Order",
    "status": "waiting_for_processing",
    "price": 10000,
    "estimated_shipping": "2022-11-27T00:00:00Z",
    "items": ["2c3fd740-6181-42ac-b95f-15ff2560ac6f"
    ]
  },
  "observers": ["104900eb-ee03-4d44-89e7-
    fbefa0ff2edc"],
  "actors": ["3aec81ab-939f-46ef-9399-e05030f3d483"]
    ,
  "important": ["activity.status", "
    activity.estimated_shipping"],
  "children": ["a22b34b2-e5ce-4eab-84c9-c9858c58b737
    "]
}
```

Fig. 1: Example for a .json which describes an activity

Fig. 1 shows an example JSON enabling inheritance through child activities while enforcing immutability after branching.

This schema ensures backward compatibility with legacy systems through XML/JSON converters and supports inheritance via child activities.

### *B. Branch Management and Immutability*

Branches in SAP's BUTLER architecture represent hierarchical groupings of atomic activities, forming directed acyclic graphs (DAGs) that model complex business processes such as Purchase Orders (POs) and Advanced Shipping Notifications (ASNs). Each branch is rooted in a parent activity (e.g., a PO) and grows through child transactions, enabling granular tracking of multi-step workflows. Key features include immutability enforcement after child creation, configurable aggregation rules for process-state derivation, and manual closure protocols to manage operational complexity.

- Branches: Logical groupings of activities structured as directed acyclic graphs (DAGs). Root nodes (e.g., POs) define aggregation rules for the entire branch.
- Immutability Enforcement: Once a node spawns children, it becomes immutable to preserve historical integrity. Modifications require the creation of new branches that reference legacy nodes for auditability.
- Size Constraints: Branches exceeding operational thresholds (e.g., 100 nested activities) are manually closed to maintain manageability.

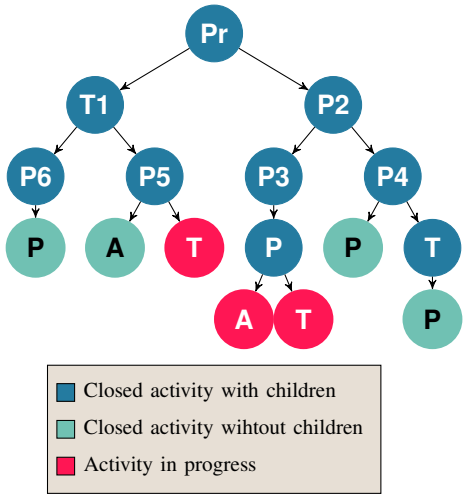


Fig. 2: Branching activities example

The provided example on Figure 2. illustrates the structural and functional characteristics of activity branches within a workflow system, demonstrating their capacity to organize diverse tasks through hierarchical relationships. As depicted in Figure 1, branches serve as logical containers for both completed and ongoing activities, where nodes represent distinct activity types with specialized roles: processes (P) handle multi-state complex operations, transactions (T) manage simplified tasks like payments with limited states (2-3), and actions (A) coordinate multi-party interactions requiring additional state transitions.

The BUTLER architecture implements a sophisticated system of node immutability and access control to maintain data integrity and manage visibility across stakeholders. The following is a summary of these rules.

*1) Immutability rules:*

- The nodes are initially mutable, allowing the changes in the attributes after creation.
- A node becomes immutable once it has any child nodes.
- The owner (creator) of a node loses the modification rights when the node becomes immutable.
- The system can still add new children to immutable nodes.
- Root nodes of branches (e.g. Purchase Orders) remain mutable until the branch is closed based on aggregation criteria.

The system maintains versioned transaction histories through branch cloning rather than overwrites. When modifying an immutable PO, users create new branches referencing original nodes, preserving both the initial state and modification. The architecture demonstrates how immutable design patterns can co-exist with dynamic business requirements.

These rules combine to create a system that preserves historical transaction integrity while enabling controlled process evolution. The architecture allows business flexibility through new branch creation rather than modifying existing immutable nodes, maintaining auditability of transaction histories.

*2) Access Control:*

- By default, branches are only visible to the root node owner.
- The ”observer” attribute allows specified users read-only access to a node and its ancestors.
- The ”actor” attribute allows users to create child nodes under a specific node.
- Observers and actors can add remarks to nodes they have access to.

*3) Visibility Rules:*

- Observers can view the assigned node, its root, and all nodes in between.
- Observers cannot modify nodes or view child nodes of their assigned node.
- Actors have similar visibility as observers, but can create new child nodes.

This system ensures data consistency while allowing flexible, role-based access control throughout the transaction lifecycle. Balances the need for immutability in completed transactions with the flexibility required for ongoing business processes.

### C. Example for immutability and observability

The system allows editing of certain node attributes. For instance, it can add new children to a node. Additionally, specific nodes, such as the root nodes of branches (e.g., PO), cannot be immutable even if they have children, unless the branch is closed based on aggregation criteria.

By default, branches are invisible to all users except the owner of the root node. However, certain node attributes can alter this behavior. Any node can have an ”observer” attribute, which is an array of user IDs. Users listed in this array have read-only access to the node and all its parents. In other words, observers can view the specific node, the root node, and all nodes between them, but they cannot modify these nodes or view any children of the specific node.

On the other hand, the ”actor” attribute provides similar functionality but allows the actor to create new children under the node. Both observers and actors can create remarks for the specific node they are associated with.

### D. Server-side implementation

The backend engine of the BUTLER architecture is responsible for enforcing business logic, managing states, and ensuring data integrity across the distributed system through:

- Schema enforcement and validation,
- State management and conflict resolution,
- On-the-fly processing,
- Aggregation between transactions,
- Gateway integration.

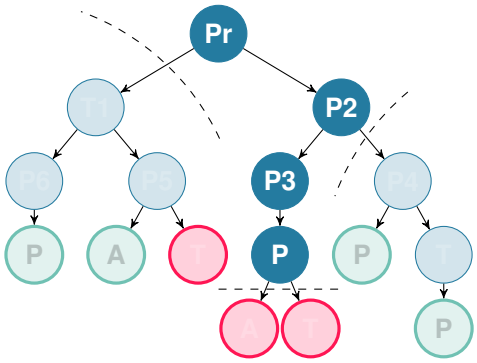


Fig. 3: Examples of visibility by observers and actors regarding node ”P” based on Fig. 2.

The backend stores JSON schemas for different activity types, which the gateway uses to validate incoming data before storage. This ensures that all activities adhere to predefined structures, maintaining consistency across the system.

The backend maintains a state machine for each activity type, enabling: State-based aggregation for calculating branch statuses, Prevention of invalid state transitions. For example, if an activity is in the ”ALMOST FINISHED” state, the backend will reject attempts to revert it to ”FAR FROM FINISHED”, preserving the logical progression of processes.

The backend provides mechanisms for aggregating states across branches, allowing for a holistic view of complex business processes. This is crucial for maintaining an accurate representation of multi-step workflows like Purchase Orders or Quotations.

While the backend is a standalone unit, it is tightly coupled with the gateway to enforce rules and logic. This coupling ensures that all data that passes through the system complies with the defined business processes and data integrity requirements.

The server-side database and backend engine implementation of BUTLER enable seamless management of complex multi-party business transactions. This implementation leverages service-oriented architectural principles to facilitate reliable, secure, and efficient transaction processing across organizational boundaries.

- Distributed Graph Database: Stores activity trees with adjacency lists for efficient traversal.
- Columnar Encoding: it could be used instead of raw JSON.
- GraphQL Optimization: Enables selective field retrieval, cutting payload sizes siginificantly compared to REST.

BUTLER’s branch management balances hierarchical flexibility with centralized control, enabling scalable tracking of intercompany transactions. By combining immutable activity trees, declarative aggregation rules, and hybrid timestamping, the architecture reduces reconciliation efforts. Future enhancements should integrate machine learning for dynamic aggregation rule generation and zero-knowledge proofs for confidential branch operations.

### E. Notification system

The Publish/Subscribe Notification System (PSNS) uses topic partitioning aligned with business units. Priorities are assigned via activity-important fields, triggering notification alerts for critical changes like payment failures.

*1) Functionality:*

- Reads observers, actors, and important fields from activity descriptions.
- Notifies involved clients when activity descriptions are updated in the database.
- Allows clients to fetch only updated fields, reducing unnecessary notifications.

*2) Optimization features:*

- Notifications include names of updated fields.
- Clients can fetch only updated fields instead of entire activities.
- Customized notification system based on client preferences.

*3) Handling prioritized updates:*

- ”Important” field in activities can hold names of critical attributes.
- When an important field is updated, PSNS:
  - Notifies the client system
  - Triggers an alarm for human operators
  - Creates an immediate report for tracking important changes

This system enables efficient updates, reduces unnecessary notifications, and ensures that critical changes are communicated promptly to human operators.

UI Features, for practical human usage:

- Branch Cloning: Creates new POs from legacy branches while preserving audit trails
- Delta Reports: Highlights changed fields since last view
- Aggregation Overrides: Manual branch closure with justification logging.

*4) PSNS example:* When an activity first enters the database, PSNS captures three key pieces of information: observers, actors, and important fields. The system operates through message brokers, which allows it to scale effectively.

PSNS notifies relevant clients whenever someone updates an activity in the database. This notification system ensures clients can always access the most current information about activities they’re involved with. Instead of sending the entire activity data, PSNS only sends the names of updated fields, allowing clients to retrieve just what changed.

The system intelligently manages notifications in several ways. By only sending information about what changed, PSNS reduces unnecessary data transfer and provides customized notifications based on client preferences. While some updates are minor and don’t require immediate attention, certain changes need prompt human awareness. The system addresses this through the ”important” field designation.

When an activity’s important fields change, PSNS does more than just notify the client system. It also triggers an alarm for the human operator.

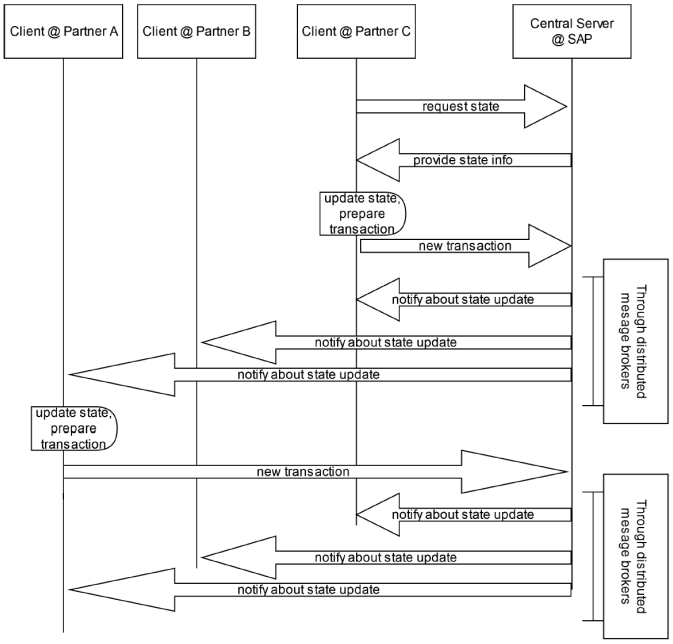


Fig. 4: An example sequence chart showing how client-side components interpret server messages as process-state updates.

This approach ensures human operators stay informed about critical changes. Important status changes and other vital updates receive immediate attention, while routine updates are handled efficiently without overwhelming users.

### F. Client-side application

Modern client-side applications must deliver optimized user experiences while balancing functional complexity with simplicity of the interface. These applications serve multiple stakeholders, including end-users, administrators, and third-party integrators, requiring careful consideration of diverse user needs.

- Multi-Workflow Navigation System - Role-based, status-driven workflows.
- Data Presentation Framework - dashboards and user interfaces, real-time data update.
- Information architecture - Notifications when next step needed by GUI user.

## VII. Use Case Validation: Purchase Order Process in SAP’s BUTLER Architecture

The BUTLER architecture redefines Purchase Order (PO) processing through activity-centric design, immutable transaction tracking, and configurable aggregation rules. By modeling each PO subprocess (e.g., RFQ, ASN, payment) as atomic activities in a hierarchical structure, BUTLER achieves unambiguous process-state visibility across stakeholders while maintaining centralized governance.

### A. Activity-Centric Process Design for PO Workflows

A Purchase Order (PO) process using BUTLER would follow these key steps:

*1) Initiation and Creation:* A new PO is created as a root node in a BUTLER branch. This node contains essential information such as items, quantities, prices, and estimated shipping dates. The PO node has an aggregation scheme defined, which determines how the overall status of the branch will be calculated.

*2) Approval and Confirmation:* The supplier approves the PO by creating a feedback node as a child of the PO node. After approval, Advanced Shipping Notification (ASN) nodes are created as children of the FB node.

*3) Processing and Updates:* If changes are needed to the PO after it has children, a new branch with a new PO node is created, referencing the previous branch as history. The backend executes the aggregation scheme after each update in the branch, computing the overall state based on predefined criteria.

*4) Notifications and Monitoring:* The Publish/Subscribe Notification System (PSNS) notifies relevant parties of updates to the PO and its child nodes. Clients can set up automations to respond to these notifications, enabling automated updates based on their internal processes.

*5) Integration with SAP Systems:* BUTLER facilitates seamless integration between SAP internal systems, legacy systems, and external partners. JSON files can be used to trigger updates when a new PO is extracted or when any status changes occur in the business process.

*6) Completion and Closure:* As goods are received and invoices are processed, corresponding nodes are added to the branch. The branch is considered complete when all required activities (e.g., full quantity received, invoice paid) are fulfilled according to the aggregation scheme. Once completed, the root branch is labeled ”DONE” and becomes immutable.

This process ensures the atomicity of events, flexible notifications configuration, and seamless integration with existing SAP systems while providing a clear, unambiguous view of PO status to all stakeholders.

## VIII. Summary

The BUTLER architecture is a collaborative project between SAP and the Budapest University of Technology and Economics (BME). Theis ”Intercompany BUsiness Transaction pLat- form architEctuRe” (aka BUTLER) provides a scalable, cloud-based framework for managing multi-party business transactions, focusing on resolving concurrency issues, ensuring process-state visibility, and maintaining centralized governance. The core feature of the BUTLER architecture is its activity-centric design, where all processes, messages, and actions are represented as atomic activities. These activities are organized into hierarchical structures called branches, enabling detailed tracking of workflows while preserving historical integrity. The system’s technical components include central server nodes that enforce transaction ordering and synchronization using distributed databases, client nodes that facilitate stakeholder interactions with role-based access controls, and a publish-subscribe notification system that efficiently distributes updates, prioritizing critical changes.

In the BUTLER architecture, nodes become *immutable* after spawning child nodes to ensure audit trails and historical integrity. Modifications require the creation of new branches referencing legacy nodes. The system’s integration capabilities allow seamless cooperation with SAP systems and traditional infrastructure (such as EDI/XML protocols), improving data flow and visibility across supply chains while adapting to changing business needs. In the Purchase Order (PO) workflow, orders are modeled as atomic activities with defined aggregation schemes, and the system supports subprocesses like POs, ASNs, payments, and completion tracking. The backend engine manages state transitions, enforces validation rules, and aggregates statuses for a holistic process view, implementing hybrid timestamping for accurate event ordering.

Overall, BUTLER provides a robust solution for intercompany business transaction management, balancing distributed execution with centralized control. It offers numerous benefits, including increased productivity through simplified transactions, transparent audit trails via immutable activity tracking, and flexibility in handling complex multi-party scenarios. Future development includes the introduction of machine learning for dynamic aggregation rule generation, and the application of zero-knowledge proofs for confidential operations.

## References


[1] Marva Dryke, Managing Director, Business Consulting, Ernst & Young LLP “EY:How to drive efficient and future-focused intercompany accounting” EY article NY, 23 Feb 2021.

[2] Ritala, Paavo and Jovanovic, Marin,“ Platformizers, Orchestrators, and Guardians: Three Types of B2B Platform Business Models” In A. Aagaard & C. Nielsen (Eds.), Business Model Innovation: Game Changers and Contemporary Issues. Palgrave Macmillan., http://dx.doi.org/10.2139/ssrn.4399864, 7 Apr 2023

[3] Hasna, Raisa & Miftahuddin, Asep. “Trends in Platform Economy Research: A Bibliometric Analysis.“ Jurnal Manajemen Sistem Informasi. (2024)

[4] Papazoglou, Michael P.”Web Services and Business Transactions”- 2003 https://doi.org/10.1023/A:1022308532661

[5] Dashkevich, N.; Counsell, S.; Destefanis, G. Blockchain Financial Statements: Innovating Financial Reporting, Accounting, and Liquidity Management. Future Internet 2024, 16, 244. https://doi.org/10.3390/fi16070244

[6] Reza Toorajipour, Pejvak Oghazi, Vahid Sohrabpour, Pankaj C. Patel, Rana Mostaghel, Block by block: A blockchain-based peer-to-peer business transaction for international trade, Technological Forecasting and Social Change, 2022, ISSN 0040-1625, https://doi.org/10.1016/j.techfore.2022.121714.

[7] Wattana Viriyasitavat, Tharwon Anuphaptrirong, Danupol Hoonsopon, When blockchain meets Internet of Things: Characteristics, challenges, and business opportunities, Journal of Industrial Information Integration,2019, https://doi.org/10.1016/j.jii.2019.05.002.

[8] Omar, Ilhaam A and Jayaraman, Raja and Debe, Mazin S and Salah, Khaled and Yaqoob, Ibrar and Omar, Mohammed: Automating procurement contracts in the healthcare supply chain using blockchain smart contracts, IEEE access,2021.

[9] Alqarni, Mohammed Ali and Alkatheiri, Mohammed Saeed and Chauhdary, Sajjad Hussain and Saleem, Sajid: Use of blockchain-based smart contracts in logistics and supply chains, MDPI, Electronics, 2023

[10] Tokkozhina, Ulpan and Martins, Ana Lúcia and Ferreira, Joao C.: Use of Blockchain Technology to Manage the Supply Chains: Comparison of Perspectives between Technology Providers and Early Industry Adopters, Journal of Theoretical and Applied Electronic Commerce Research, 2022, https://www.mdpi.com/0718-1876/17/4/82

[11] Lamport, L. (2019). ”Time, clocks, and the ordering of events in a distributed system. Concurrency: The Works of Leslie Lamport.” doi:10.1145/3335772.3335934