TABS+: Transforming Automatically BPMN Models to Smart Contracts with Nested Collaborative Transactions

Transforming BPMN Models to Smart Contracts with Support of Nested Transactions

Christian, G., Liu

Faculty of Computer Science, Dalhousie University, Chris.Liu@dal.ca

Peter Bodorik

Faculty of Computer Science, Dalhousie University, Peter.Bodorik@dal.ca

Dawn Jutla

Sobey School of Business, Saint Mary's University, Dawn.Jutla@gmail.com

Development of blockchain smart contracts is more difficult than mainstream software development because the underlying blockchain infrastructure poses additional complexity. To ease the developer's task of writing smart contract, as other research efforts, we also use Business Process Model and Notation (BPMN) modeling to describe application requirements for trade of goods and services and then transform automatically the BPMN model into the methods of a smart contract. In our previous research we described our ***TABS*** approach and tool to ***T****ransform* ***A****utomatically* ***B****PMN models into* ***S****mart contracts* that also supports sidechain processing. In this paper, we describe how the TABS approach is augmented with the support of a BPMN collaborative transaction by several actors, which is an extension of the BPMN concept of a transaction that can be defined only on activities of a single actor. Our approach analyzes the BPMN model to determine which patterns in the BPMN model are suitable for use as collaborative transactions. The found BPMN patterns that are suitable as transactions are shown to the developer who decides which ones should be deployed as collaborative transactions. We describe how the nested BPMN collaborative transactions are supported by an automatically created transaction mechanism. We also overview the TABS+ tool, built as a proof of concept, to show that our approach is feasible. Finally, we provide estimates on the cost of supporting the nested BPMN collaborative transactions.

CCS CONCEPTS • Information systems → Data management systems → Database management system engines → Blockchain databases • Software and its engineering → Software notations and tools → Formal methods • Computer systems organization → Architectures • Information systems → Information systems applications • Computing methodologies → Modeling and simulation

Additional Keywords and Phrases: Blockchain, Business Processes Modeling Notation (BPMN), Automated Generation of Smart Contracts, Transforming BPMN Models to Smart Contracts, Discrete Event (DE) Modeling, Finite State Machine (FSM), Hierarchical State Machines (HSM), Privacy, Trade of Goods and Services, Transactions, Long-term Collaborative Multi-method Transactions, Nested Transactions, Sidechain, Privacy, Optimistic Methods

ACM Reference Format: Automatically generated.

1 INTRODUCTION

Introduction of the Bitcoin cryptocurrency [Nakamoto 2008] and its subsequent rise [Marr 2017] spurred great interest in cryptocurrencies and blockchains. Further blockchain platforms, such as Hyperledger fabric and Ethereum, followed the Bitcoin blockchain and many of them succeeded while many have failed. The concept of a smart contract [Szabo 1997, 2018; Buterin 2015] emerged as a collection of methods written in a Turing-complete high-level language, which is also stored on the ledger and thus benefits from the protection provided by blockchains. In addition, the blockchain infrastructure utilizes cryptographic concepts and methods to provide the blockchain's desirable properties of trust, immutability, availability, and transparency, amongst others. However, as a new technology, blockchain and their smart contracts pose new challenges in both the blockchain infrastructure and in developing smart contracts and applications that use them. Thus, in addition to tackling the blockchain infrastructure issues of scalability, transaction throughput, and high costs, the development of smart contracts also received much attention by the research and development communities as can be seen by many literature surveys on the topic, such as [Taylor 2019, Khan 2021, Vacca 2021, Belchior 2021, Saito 2016, Garcia-Garcia 2020, Lauster 2020, Levasseur 2021].

In [Khan 2021], research on blockchain smart contracts is classified into the categories of improvement and usage, in which the improvement category includes research and tools to improve smart contracts either when writing them or once they exist. Improvement approaches for writing smart contracts generally rely on defining a programming language that facilitates creation of smart contract with desirable properties. However, although these approaches lead to smart contract programs with formally proven safety properties, they have not been adopted in practice as they exhibit, or are perceived to exhibit by software developers, complexity that deters their adoption by developers who ultimately decide how well-accepted a programming language is. To alleviate such issues, some researchers use the well-known and used Business Process Management Notation (BPMN) to model the application requirements and transform a BPMN model automatically into the methods of a smart contract(s) [Weber 2016, López-Pintado 2019, Tran 2018, López-Pintado 2019, Bodorik 2022, Liu 2021a].

1.1 Motivation

Trade of goods and services, as opposed to securities, includes collaborative activities that arise with a sale, trade, or finance of goods and/or services between two or more parties. Furthermore, for higher priced sale items, such as combines or other special machinery, in addition to the buyer and seller parties, banks are typically involved to provide financial instruments supporting the purchase, such as a letter of credit that guarantees the payment of goods to the seller's bank upon receiving the goods or services. In addition, other parties may be involved in the trade, such as transporters/shippers for the delivery of the purchased goods, insurance companies for risk mitigation, and regulators if the goods are crossing international or state/provincial/municipal boundaries. Thus, trade transactions involve many parties that collaborate in activities that may be long term, e.g., spanning, minutes, hours, or days, as the collaborative activates may involve different software or human events that trigger invocation of the smart contract methods by a distributed application.

When writing smart contracts that support such long-term collaborations, also referred to as collaborative transactions, such collaboration is conveniently represented as a collection of smart contract methods, but it is awkward and difficult to represent it as one smart contract method. The problem is that a blockchain supports only a transaction that contains ledger updates made by an execution of any one of the methods of the smart contract. And how to support a blockchain transaction that spans execution of a multiple methods of a smart contract is one of the issues that we addressed in [Liu 2023]. In that research report we provide the developer with the ability to define a ***m***ulti-***m***ethod (***mm***) transaction as a collection of smart contract methods that are *independent,* i.e., that is a subset of the smart contract methods that do not invoke methods or

refer to objects that are not declared within the context of that subset of the transaction methods. We also describe how to use pattern augmentation of smart contract methods to automatically create mechanisms that support the mm-transactions with the properties of ***A***tomicity, ***C***onsistency, ***I***solation, and ***D***urability, also referred to as ***ACID*** properties, that are defined for blockchains in a manner analogous to the corresponding ACID properties in DB systems. However, although the concept of the mm-transaction supports the use of smart contracts with such transactions, it further complicates the design and development of a smart contract. The developer needs to analyze the application requirements to determine the collaborative long-term transactions that need to be supported first and then build the smart contract and the **d**istributed **app**lication (***Dapp***) around those transactions – thus further complicate the design and development of a smart contract.

To ease the development of the smart contract, as in other research [Weber 2016, López-Pintado 2019, Tran 2018, López-Pintado 2019, Mendling 2018, Loukil 2021], we developed a methodology that starts with using a well-known BPMN to model the application requirements and then transform the BPMN model automatically into the methods of a smart contract(s). Our approach to the transformation is quite different from the other research and will be described in further details in the next section. In [Bodorik 2022, Liu 2021b, Liu 2021c, Liu 2022], we describe our approach, and the ***TABS*** tool (***T***ool to ***A***utomatically Transform a ***B***PMN Model to ***S***mart Contract Methods) as a ***P***roof ***o***f ***C***oncept (***PoC***), to show the feasibility of our approach. In addition to transforming a BPMN model to a smart contract, the TABS approach and tool also support sidechain processing, wherein the developer[1] may choose certain BPMN patterns to be processed on a sidechain in a form of a smart contract deployed on a sidechain - this is advantageous cost-wise when sidechain processing is much cheaper than the mainchain processing.

In [Bodorik, 2022], to select which BPMN patterns should be processed on a sidechain, a ***D***irected ***A***ccess ***G***raph (***DAG***) representation of the BPMN model is analyzed to find ***S****ingle-**E**ntry* and ***S****ingle-**E**xit* (***SESE***) subgraphs, as a SESE subgraph was shown to be good candidate for sidechain processing due to its localized nature of processing. The edges of the graph represent the flow as data is being processed in possibly concurrent streams of execution. A SESE subgraph is such that all its vertices have edges connecting it only to the vertices of the subgraph. The only exceptions are two nodes of the subgraph, a *Single Entry* and *Single Exit* vertices. The *Single Entry* vertex contains incoming edge(s) from the vertices external to the graph. These external incoming edges are in addition to the other internal edges that connect the *Single Entry* vertex to the other nodes in the subgraph. Similarly, the *Single Exit* vertex contains edges to the vertices external to the graph. These external incoming edges are in addition to the other internal edges that connect the *Single Entry* vertex to the other nodes of the subgraph. Because there is only one entry and one exit point, once the flow of execution enters the subgraphs, it remains within the subgraph until the flow of execution leaves via the exit vertex.

In this paper we address the problem of supporting long-term collaborative transactions by the TABS approach when transforming a BPMN model into a smart contract. There are two main issues that need to be addressed. The first one is similar to the issue of a blockchain limiting its transaction to be the result of an execution of any single method of a smart contract. The BPMN specification limits the concept of an atomic transaction to a sub-process that is executed within the context of one actor. That is, the BPMN specification does not provide a concept of a transaction, with its all-or-nothing properties, that is a collaboration of several actors. Our research discovered that our approach of using the SESE properties to support the side-chain processing can also be exploited in defining BPMN collaborative transactions within a BPMN model and in creating a mechanism that supports the transactional properties for such transactions.

[1] We use the term developer also to refer to the developer of BPMN models who is also referred to in the context of BPMN models as a modeler.

1.2 Goal and Objectives

The main goal of this paper is to describe how the TABS approach was augmented to support defining and supporting the collaborative transactions and how the TABS tool is augmented into the **TABS+** tool that also provides the mechanism to support the BPMN collaborative transactions that may be nested. The specific objectives include:

- Showing that the result of transforming a BPMN model into the ***D**iscrete **E**vent (**DE**)* – ***F**inite **S**tate **M**achine (**FSM**)* model (***DE-FSM*** model) can be analyzed to find patterns that are suitable candidates for collaborative transactions that may be nested.
- Showing how TABS+ provides an automated transactional support for nested BPMN collaborative transactions.
- Showing how the concept of a collaborative transaction may be extended to represent nesting of such transactions and how the transactional mechanism may be created automatically to support the nested collaborative transactions.
- Develop a PoC that our approach is feasible by incorporating the support of nested collaborative transactions into the TABS tool, thus creating TABS+ tool.

Provide an estimate of overhead in supporting the nested collaborative BPMN transaction using the TABS+ approach.

1.3 Inserting CCS concepts

It was already reported above that the TABS approach, which provides for an automated transformation of a BPMN model into the methods of a smart contract with the support for sidechain processing, has been presented in [Bodorik 2022] with earlier work described in [Bodorik 2021, Liu 2021a, Liu 2021b, Liu 2021c, Liu 2022]. In this paper we tie these concepts together and describe how the TABS approach is amended to support transformation of BPMN models into smart contracts with support for the nested BPMN collaborative transactions, resulting in the TABS+ approach and tool. More specifically:

- We show that the transformation of a BPMN model into DE-HSM model and then DE-FMS model allows for the analysis of the model's graph representation to find patterns, SESE subgraphs in the DE-FSM model, that have localization of execution properties that make them suitable candidates for BPMN collaborative transactions.
- We augment the TABS tool (into TABS+) and its approach to support BPMN collaborative transactions that are transformed into smart contracts that contain transactions that *span multiple methods* and are *nested*.
- We report on costs associated with creation of a transaction mechanism that supports the nested collaborative BPMN transactions.
- More importantly:
- We show that the design of Dapps to support applications in the vertical of trade of goods and services is greatly simplified using our approach.
    - Instead of the developer first needing to identify and design transactions that need to be supported and then develop the methods of the smart contract that supports those transactions, the developer uses BPMN modeling to express the collaboration of actors to achieve the intended trade of goods and services, which also includes finance, and then uses the TABS+ tool to analyze the BPMN model to identify SESE subgraphs that serve as potential transactions that the application should support.
    - The developer lets the system know which of the SESE subgraphs should form the transactions that may be multi-method nested. The system provides the developer with an estimate on the overhead cost incurred by the transactional mechanism that system provides in automated fashion.
    - The developer also lets the system know which of the SESE subgraphs should be deployed and executed on a sidechain.

1.4 Related Work

Closest to our work is research in [Weber 2016, López-Pintado 2019, Tran 2018, López-Pintado 2019, Mendling 2018, Loukil 2021] that was discussed already in a previous subsection. It deals with transformation of a BPMN model into a smart contract that contains a set of tasks that are executed under the control of a monitor. The monitor and tasks are a part of the system deployed in a form of a smart contract containing a monitor/mediator that controls the collaboration and execution of tasks. The off-chain part of the system transforms application's input into calls to the methods of the smart contract. Input is parsed to determine application's input and choreography is used to determine which task is to be executed. Incorrect sequences of input form "non-conformant" input streams that need to be identified by the system, while correct sequences should result in execution of correct tasks. Below we briefly overview how our previous work on blockchain smart contracts relates to this paper:

- [Bodorik 2021, Liu 2021b] describe how an application, described using an FSM, can be transformed automatically into the methods of smart contract such that sidechain processing is supported. We describe the transformation process to derive the methods of the smart contract and the system architecture that includes a bridge between the mainchain and the sidechain to support the sidechain processing.
- [Liu 2022] describe our early work on using DE-HSM models for multi-modal modeling when transforming a BPMN model to the methods of a smart contract.
- [Bodorik 2022] formalizes the approach of the previous work to transforming an application expressed as a BPMN model into the methods of a smart contract. The results of [Liu 2021b, Liu 2022] are used in formulating the transformations in design phase, while we use the results of [Bodorik 2021] in supporting sidechain processing.
- [Liu 2022] raises the issue of how to represent and support BPMN long-term collaborative transactions.
- Finally, [Liu 2023] describes a mechanism for a developer to specify blockchain transactions that span executions of multiple methods of a smart contract and thus extend the native blockchain mechanism that supports a transaction as a result of executing a single smart contract method. In addition, the paper also describes how pattern augmentation technique is used to automatically create a transactional mechanism for the multi-method transactions specified by the developer.

This paper ties together our previous work on transforming a BPMN model into methods of a smart contract(s) by describing how the BPMN collaborative transactions are supported and how nesting of such transactions is supported. We also describe the TABS+ tool as a PoC that our approach to transforming BPMN models into smart contract methods with the support of nested BPMN collaborative transactions and sidechain processing.

It should be noted that our approach uses SESE graphs of a DAG representation of a BPMN model not only in defining the BPMN nested transactions as described in this paper, but that they are also used tin defining which patterns of a BPMN model are suitable for processing on a sidechain as described in [Bodorik 2022]. The same approach is used as in both cases because of the localization property of SESE subgraphs. When supporting sidechain processing, the localization property of a SESE subgraph is suitable for sidechain processing as the localized logic on a sidechain incurs less overhead when moved to the sidechain for processing.

The localization property of SESE subgraphs is also exploited for finding BPMN patterns that are suitable for defining collaborative transactions. However, the reason for exploiting the localization property of the SESE subgraphs is not the cost of execution, but rather the isolation property that is achieved: The effects of the transaction execution is localized and if the transaction is aborted, it will not affect the rest of the smart contract due to its localization of execution.

Because the support for sidechain processing and for supporting BPMN collaborative transactions is based on the SESE subgraphs, transformation up to and including finding SESE subgraphs are the same and hence in our description that follows we also refer to material and use examples, such as figures, that were already described in [Bodorik 2022].

1.5 Outline

Section 2 overviews the background information for this paper. It briefly overviews ***H***ierarchical ***S***tate ***M***achines (***HSM***s) and multi-modal modeling. Considering that the support of the nested collaborative transactions is based on the TABS approach and tool, the background section also overviews the TABS approach to automated transformation of a BPMN model into the methods of smart contracts with the support of sidechain processing.

Section 3 reviews the main features of transformations from BPMN to smart contract methods and then shows how the approach to transformations and the TABS tool are augmented to support defining BPMN collaborative transactions for a BPMN model and to create a mechanism to support the collaborative transaction in addition to supporting the sidechain processing. Section 4 then addresses how the support of nested transactions is achieved.

Section 5 presents the description of the TABS+ tool and cost estimation for the support of the nested collaborative transaction. The final section reviews related work and provides conclusions and suggestions for future work.

2 BACKGROUND

We first briefly overview BPMN modeling and describe a simple BPMN use case that had been used also by other research in reports dealing with BPMN model transformation into a smart contract. We then overview HSMs and DE-HSM multi-modal modeling that combines DE modeling for concurrency with HSM/FSM modeling for functionality. This is followed by a high-level overview of the transformations performed in the TABS.

2.1 BPMN – Business Process Management Notation

BPMN was developed by the OMG organization [Business Process Model and Notation (BPMN), Version 2.0. 2023] with the objective of BPMN models to be understandable by all business users, from business analysts, through technical developers implementing the processes, to people managing those processes. It is viewed as a de-facto standard for describing business processes. That it has been adopted in practice is demonstrated by the many software platforms available that provide for modeling of business applications with the objective to automatically create an executable application from the BPMN model. For instance, Oracle Corporation uses BPMN to describe an application and transform into a blueprint of processes expressed in an executable Business Process Execution Language (BPEL) [Dikmans 2008], wherein the blueprint represents the logic of the application in terms of concurrent processes and their interactions, while details of individual tasks are supplied by implementors. Another example is Camunda software platform that is also used to develop a BPMN model that is then transformed automatically into a Java application [Deehan 2021].

BPMN standard categorizes the BPMN elements into five basic categories, wherein each category may have sub-categories. As there are many good introductions and overviews in the literature, and we overviewed BPMN in some detail in [Elsevier journal], we concentrate on the description of the BPMN use case, shown in Figure 1, that has been used in several research reports on transforming BPMN models into smart contracts, for instance in [Weber 2016, Tran 2018, López-Pintado 2019] and that we also use in this paper.

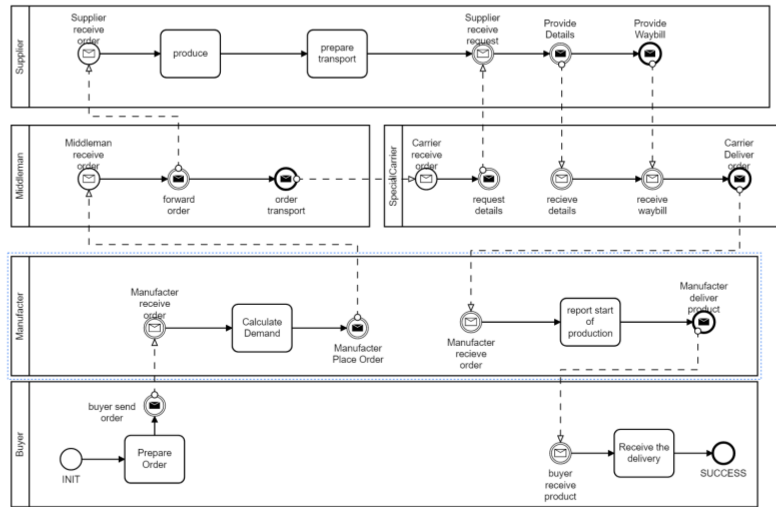


Fig. 1. A BPMN model for a supply-chain example (adopted from [Liu 2023])

This use case was adopted from [Weber 2016] to be used as a sample use case. The supply-chain use case begins with the buyer issuing a new order. Once the manufacturer receives the order, she/he calculates demand and places an order with a middleman. Middleman concurrently sends the order to the supplier and orders transport from the carrier. The producer fabricates the product and prepares it for transport. Carrier, upon receiving the request from the middleman, requests details from the supplier. Supplier provides the details to the carrier and then it prepares and provides the waybill to the carrier. Upon receiving details about the product and the waybill, both from the supplier, the carrier delivers the order to the manufacturer. Upon receiving the order, the manufacturer starts the production and when that is finished, it delivers the product to the buyer, who receives the order.

There are five actors (Buyer, Manufacturer, Middleman, Special Carrier, and Supplier), with each actor having a pool, or a swimlane within a pool, that is represented by a large rectangle representing processing performed by the actor. Arrows, represent connecting objects, which are either sequence flows or message flows. Only message flows may cross pools. It should be noted that some of the BPMNM elements represent tasks that appear in the diagram as smaller rectangles with rounded corners. A task is an activity, which is a script provided by the developer, that is executed when the flow of execution, as represented by arrows and other elements, reaches the task. For each task, the developer provides the code. The remaining BPMN elements are used to represent the coordination of the collaborative activities as performed by the actors. Furthermore, when two or more arrows exit out of an element, the element serves as a split/fork/diverging gateway, such that each arrow represents a concurrent stream of activities performed by the actor. When more than one arrow end at a BPMN element, that element serves as a join/merge/converge gateway representing joining of concurrent processing streams into one stream. Split gateways may be inclusive or exclusive, wherein an exclusive gateway allows only one stream to leave, while inclusive may allow many streams to proceed with execution concurrently. An inclusive joining gateway passes a token only if all incoming paths are enabled. Both splitting and merging gateways may have a condition specified on each path.

2.2 FSMs, Hierarchical State Machines, and Multi-modal Modeling

Because FSM modeling has been used frequently in the design and implementation of software, the FSM modeling has been expanded with features, such as a guard along an FSM transition to specify a Boolean condition on the state's variables that must evaluate to true for the transition to take place. In the late 80's, FSMs were extended to address the issues of reuse of patterns with the concept of hierarchy, leading to Hierarchical State machines (HSMs) that may contain states that are themselves other FSMs. Harel (1987) showed that FSMs can be combined hierarchically: A single hierarchical state at one level can be considered to be in several states concurrently as represented by an FSM(s) in a lower level of the hierarchy, and FSMs may also be combined leading to concurrent FSMs.

An HSM can be defined using induction as follows (due to [Harrel 1987] and described in [Girault 1999], [Yannakakis 2000], and others): In the base case, an FSM is a hierarchical machine. Suppose that M is a set of HSMs. If F is an FSM with a set of states, S, and there is a mapping function f: S->M, then the triple (F, M, f) is an HSM. Each state, s ∊ S, that represents an HSM is replaced by its mapping (f(s)). HSMs recognize the same language as its corresponding flattened FSM. HSMs do not increase expressiveness of FSMs, only succinctness in representing them.

Girault et al. (1999) describe how HSM modeling can be combined with concurrency semantics of a number of several models, including communicating sequential processes [Hoare 1987] and discrete events [Cassandras 1993]. Girault et al. (1999) describe how an HSM model can represent a module of a system under a concurrency model that is applicable only if the system is in that state. This enables representation of a subsystem using a particular concurrency model that may be nested within a hierarchical state of a higher-level FSM. This may be used in multi-modal modeling, in which different (hierarchical) states may be combined with different concurrency models that are best suitable for modelling of concurrent activities for that particular state. We exploit the concept of multi-modal modeling to allow the designer to model concurrent, but independent activities, by concurrent FSMs at the lower level of hierarchy. In [Liu 2022], we showed that a multi-modal model that combines Discrete Event (DE) modeling with FSM modeling may be used to model trade finance applications – such a combination of models is referred to as DE-HSM multi-modal modeling. We also showed that it is possible to automatically transform a DE-HSM model of such an application into methods of a smart contract deployed on a blockchain.

A DE-HSM has external inputs, and it produces outputs. The model represents how external inputs form inputs to the sub-models and how those sub-models are interconnected to produce the final output. However, if the sub-model's interconnection is such that there are no loops, then the model can be viewed as a zero-delay model [Yannakakis 2000] in which the individual DE queues may be combined into one DE queue.

Feedback loops in a DE-HSM arise due to feedback loops in the BPMN model from which the DE-HSM model is derived. None of the approaches, to producing smart contract from a BPMN model that were discussed in the introductory section, addressed the issues of the feedback loops in a BPMN model. Feedback loops cause various difficulties, such as deterministic semantics, and will be discussed further in a later section. When we use the term DE-FSM model, we refer to a model that does not contain hierarchical states.

2.3 TABS Approach to Transforming BPMN Models to Smart Contracts

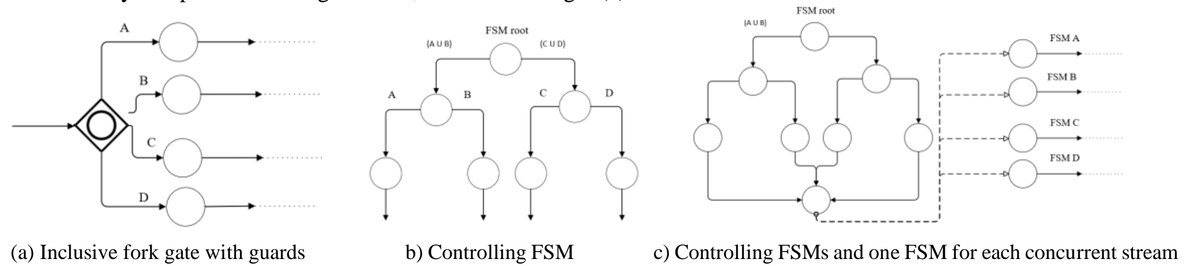
To alleviate difficulties in writing smart contracts, many researchers represent the Dapp requirements using a BPMN model as a starting point to model the application requirements. We also start with a BPMN model, but unlike the other approaches that transform the BPMN model directly into the methods of a smart contract, we use several transformations that result in a system represented as an interconnection of subsystems, with each subsystem represented using multi-modal modeling.

2.3.1 DE-HSM and DE-FSM Modeling to Represent Concurrency and Functionality

In multi-modal modeling we use, the functionality is represented using an HSM, while concurrency is modeled using Discrete Event (DE) modeling - thus resulting in a multi-modal DE-HSM model. Furthermore, each HSM model may be an interconnection of further DE-HSM sub-models [Bodorik 2021, Bodorik 2022, Liu 2022]. Eventually, by elaborating each HSM with its internal representation, the system may be "flattened out" into an interconnection of DE-FSM sub-models, in which concurrency is represented using DE modeling, while functionality is represented by an FSM.

The DE-FSM model represents flow of execution throughout the model such that many concurrent streams of executions are possible. From the BPMN start element, the flow of execution is forked using an inclusive gateway into one of n flows, one for each of the n actors, while the functionality of each flow is expressed using an FSM derived for each actor. Of course, there are likely to be forking gateways in the BPMN model that may result in forking a flow of execution into multiple executions of concurrent flows. And similarly, there are likely to be joining gateways in the BPMN model that join several execution-flows into one. Forking of execution flows is supported by using the concurrent FSMs.

Consider a forking inclusive gateway with guards A, B, C, and D as shown in Fig. 2(a). Outgoing edges of a BPMN inclusive fork gate represent the flows of executions/computations of multiple concurrent streams. The flow of execution for such a gate within a sub-model is represented by a control FSM, shown in Fig 2(b) that evaluates the guards for multiple concurrent streams and results in four flows of executions, in which concurrency is expressed using a DE model, while functionality is represented using an FSM, as shown in Fig. 2(c).



(a) Inclusive fork gate with guards b) Controlling FSM c) Controlling FSMs and one FSM for each concurrent stream

Fig. 2. Inclusive fork gate that results in a control FSM and one FSM for each concurrent process streams (adopted from [Liu 2023])

2.3.2 Transforming a DE-FSM Model into the Methods of a Smart Contract

So how are the smart contract methods formed? Each actor has a smart contract method with the same script, wherein a smart contract method provides the execution environment for an actor as represented using the DE-FSM modeling. In other words, a smart contract method acts as an interpreter for executing the DE-FSM model that is stored within the monitor. Although the monitor script is the same across the methods of the smart contract, one method for each of the actors, the DE-FSM models are unique for each actor as they represent the actor's activities. DE modeling uses a DE queue, a priority queue of events timestamped using the global clock supporting the Dapp and the smart contract. As the Dapp executes, it takes input from the actors, i.e., from the users or their systems acting on their behalf. The user input is examined to determine the event origin in terms of the BPMN diagram. This is then used to identify which of the actors is the event's target and the actor's smart contract method is invoked with the user input passed to the method as an input parameter.

The actor's smart contract method examines the origin of the input parameter in terms of the BPMN model, and it queues the event for execution into the DE queue of events for that sub-model. Once the event reaches the head of the queue and is dequeued and "processed" by the method. The method examines the user input and determines to which of the concurrent FSMs for the actor's flow of execution it applies, and then it forwards the user input to that FSM. The FSM examines its current state and the input and fires, while producing a new state and generating its output. The output and

the new state are examined and then the method either queues another new event into the DE queue or into a DE queue of another actor, and then the process repeats until the queue is empty.

It should be noted that as there are no feedback loops within the DE-HSM and DE-FSM sub-models, all sub-models may share a single queue of events in DE modeling. Any input generated by the Dapp is first augmented with the information on the BPMN context (which BPMN element caused the method invocation) and is queued into the DE queue of events. The queued event eventually reaches the head of the DE queue and is dequeued and processed by invoking an FSM that results in another event being queued, or a task being executed or both and the processes repeats with another DE queue element dequeued and processed. For instance, if the input was for an FSM that controls outgoing streams of an inclusive fork gate, such as the one shown in Fig. 2(b), once the controlling FSM fires, its output is used to generate an event for each of the guards A, B, C, or D that was evaluated to true. However, the output from an FSM firing may also indicate that a specific task, corresponding to the BPMN task element, needs to be executed.

Fig. 3 shows the system architecture overview for the design phase and the execution phase.

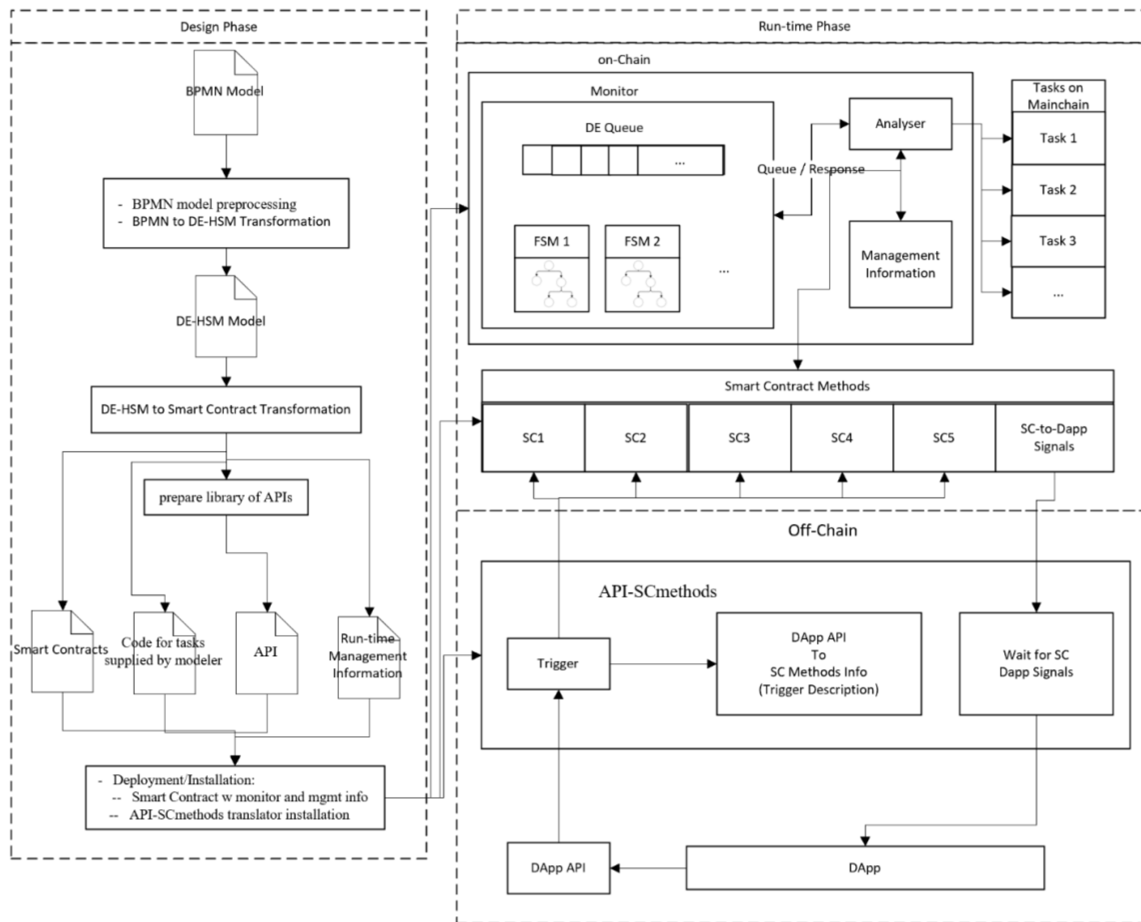


Fig. 3. Architecture Overview (adopted from [Bodorik 2022])

Referring to Fig. 3, in the design phase:

- The BPMN model is transformed into a DAG representation and then into an interconnection of DE-HSM sub-models and eventually into an interconnection of DE-FSM sub-models.
- It is then transformed into the methods of the smart contract.
- The smart contract with its methods is deployed as is an API interface that is invoked by the Dapp. The API interface examines the Dapp's input parameters and invokes an appropriate smart contract method with an input parameter containing the information about the Dapp's API call.

Execution phase includes two sub-phases: (1) Initialization and (2) Execution. Initialization deals with issues, such as authentication of the actors. Once the smart contract is deployed, execution is managed by the Monitor module. As long as the DE queue is not empty, the monitor repeatedly dequeues an element and examines it to determine which of the smart contract methods it belongs. It then invokes that smart contract method that then determines which of its FSMs should be used to process the element and it invokes that FSM to fire with input from generated from the dequeued element. FSM fires, which results in a new FSM state and output from the firing. The FSM's output and new state are examined in order to determine if a new DE element needs to be queued or if a task needs to be executed as a part of FSM firing cycle.

The above description is a high-level overview of the TABS system architecture. It should be noted that many details have been omitted, including details how sidechain processing is supported by the system. For further details, see [Bodorik 2022].

3 TABS+: TRANSFORMING A BPMN MODEL TO SMART CONTRACT METHODS WITH THE SUPPORT FOR COLLABORATIVE BPMN TRANSACTIONS

In the first subsection, we review the major aspects of transforming a BPMN model into the methods of a smart contract as described in [Bodorik 2022]. In the second subsection, we highlight how the graph representation of the multi-modal model is analyzed to find SESE subgraphs. We argue that a collaborative transaction in a BPMN model needs to be such that the transaction's activities do not export any information outside the transaction, and that such a localization property is exhibited by SESE subgraphs and hence they are good candidates for specifying a transaction at a BPMN level as a collaboration of activities by different actors. Note that the BPMN specification does not include such a transaction construct as, in BPMN, a transaction may be defined only on a sub-process that is executed within the context of a single actor and hence cannot be used in supporting collaborations of several actors.

Our approach finds all SESE subgraphs within the DE-FSM model and then transform the subgraphs back to the BPMN model representation that is shown to the developer who decides which of the SESE subgraphs should be transformed to transactions that satisfy the ACID properties.

Furthermore, when the BPMN model is transformed into the methods of a smart contract, the activities for each of the SESE subgraphs selected by the developer as a transaction are transformed into a separate smart contract method and thus satisfying the *independence* property as defined in [Liu 2023]. Consequently, transactional mechanisms described in [Liu 2023] for mm-transaction support are applied on the methods generated from the SESE subgraphs selected by the developer.

The following subsections describe the overall process of transforming a BPMN model into the methods of a smart contract under the developer's guidance with the support for specifying nested collaborative BPMN transactions together with the description how the transactional support for the BPMN collaborative transactions, even when nested to support sub-transactions, is achieved.

3.1 TABS Transformations of a BPMN Model into Smart Contract Methods

We outline the TABS transformations, of a BPMN model into the methods of a smart contract, as presented in [Elsevier paper] while omitting many details: The subsections of 3.1 describe the following:

- 3.1.1 describes the major assumptions made on the BPMN model;
- 3.1.2 presents pre-processing and forming DAG representation of the BPMN model;
- 3.1.3 describes how a DAG model is transformed into the multi-modal DE-HSM and then DE-FSM models;
- 3.1.4 describes the transforming of the DE-FSM model into the methods of a smart contract(s).

3.1.1 Assumptions

Here, we briefly highlight the assumptions made in [Bodorik 2022] on a BPMN model and its transformation into a smart contract:

Assumptions on BPMN looping and parallel construct: As in other work on transforming the BPMN model into a smart contract, we limit the semantics of BPMN looping and parallel constructs so that the models are deterministic, and we handle these constructs as special cases in the final phase of the generation of the methods of smart contracts. Removal of loops from the BPMN models facilitates the use of multimodal modeling in which DE modeling is used for concurrent execution, while functionality is expressed using either HSM or FSM modeling.

Assumptions on BPMN converging gateways: We make a simplification for an inclusive converging gateway in that we simply pass the token - we check the other pathways neither for enablement nor for a token arrival.

No mm-transaction: Multi-method transactions are not supported in the TABS tool [Bodorik 2022].

Coverage of BPMN: Not all BPMN symbols are supported. The list of currently supported symbols can be found in [Bodorik 2022].

Task element: The BPMN task element represents a self-contained task. The transformations prepare the method skeleton for each task element, and a task is invoked according to the BPMN model description. However, the code/script for the task must be supplied by the developer. This approach is standard in creating applications from a BPMN model. For instance, Oracle Corporation uses the same approach in its Oracle Business Process Analysis Suite when transforming a BPMN diagram into a blueprint for executable Business Process Execution Language (BPEL) processes [Dikmans 2008] and a similar statement also applies to the Camunda platform that also asks the modeler to provide the script for each of the PBMN model's tasks.

Off-chain storage: We adopt the standard practice to store large objects/data off-chain and store on the mainchain only the hash-code of the data stored off-chain. Any time the object is retrieved by a smart contract, to ensure the immutability property of an object stored off-chain, its hash-code is recalculated and checked against the hash code stored on the blockchain. We facilitate off-chain storage by using IPFS, which is a distributed system for storing and accessing files, websites, applications, and data [Steichen 2018]. IPFS storage is content addressable using the hash-code of the object. Storage in IPFS may be replicated and by controlling replication desirable resilience to storage failures may be achieved [Obe 2022, Nabben 2022].

*3.1.2 BPMN Pre-processing and **D**irected **A**cyclic **G**raph (**DAG**) Generation*

Once a BPMN model is created and expressed using its standard XML representation [BPMN 2.0 Introduction · Flowable Open Source Documentation, 2022], the model is transformed into an equivalent BPMN model that is well-formed, wherein the well-formed BPMN model is defined as in [Dijkman 2008]: (i) there is only one *start* event; (ii) there is only one *end* event; (iv) *fork/split decision gateways* have one incoming flow and more than one outgoing flow (v) *join/merge*

gateways have one outgoing flow and more than one incoming flow; and (vi) there are *no data-based splits/forks or joins/merges* – they are expressed using equivalent gateways constructs. The above conditions are not restrictive as any BPMN model can be transformed into an equivalent well-formed BPMN model as described in [Dijkman 2008]. Furthermore, the execution flow from the *start event* BPMN element is immediately forked, by an inclusive fork gateway, into n concurrent/parallel flows of executions, one for each of the actors. Similarly, the parallel streams for the actors are eventually joined, by a join gateway to end at vertex corresponding to the *end event* BPMN element.

Following the preprocessing, the BPMN model is transformed into its DAG representation in which the edges represent the flows of execution. Using BPMN association data, the developer annotates which data flow along the edges. Vertices represent either self-contained tasks, for which the code is supplied by the developer, and which consume the data along the incoming edger and output data on the outgoing edge, or elements that represent the emitters or receptors of information/events, such as gateways controlling the flow of execution or event generation/consumption, which also control or alter the flow of execution. Flows of executions may be concurrent as controlled by BPMN elements, such as gateways or event generators/receptors. As the BPMN model does not have looping under the stated assumptions, the graph representation of the BPMN model does not have any cycles either – hence a DAG representation.

3.1.3 DAG Transformation into DE-HSM Sub-models and then to DE-FSM Sub-models

We analyze the DAG representation of the BPMN model in order to find all SESE subgraphs. We show that due to the underlying graph being a DAG, the SESE subgraphs are also acyclic. As a SESE subgraph is acyclic, the subgraph may be analyzed using multi-modal modeling in which concurrency is modeled using DE modeling and functionality is expressed using HSMs. Furthermore, we show in [Bodorik 2022] that the DAG representation of the BPMN model can be represented as a collection of mutually exclusive SESE subgraphs, such that all DAG nodes appear exactly in one of the subgraphs and each edge either appears in one of the subgraphs or it connects two of the subgraphs. The SESE subgraphs have further properties that we discuss later when describing how the transactional mechanism is implemented to support BPMN collaborative transactions.

Each of the DE-HSM sub-models is "flattened out" by representing each HSM by its representation as an interconnection of its DE-HSM sub-models until each off the sub-models is a DE-FSM model. It is the DE-FSM model representation that is used to produce the methods of a smart contract(s).

3.1.4 Transforming DE-FSM Model into the Methods of a Smart Contract(s)

Each actor has a smart contract method that acts as a monitor and controls the flow of execution for that actor. The flow of execution is represented by a DE queue of events that are continuously dequeued during the execution phase by the monitor and processed by taking the dequeued element's information to determine which FSM should be fired with input derived from the dequeued DE element as described in the subsection 2.3. Once the smart contract(s) is generated, compiled and deployed, initialization is performed, and the smart contract is ready for invocation of its methods.

3.2 BPMN Collaborative Transactions: Properties and Specifying

Transformations described above are for the case when there are no BPMN collaborative transactions, and no sidechain processing is used. Before we discuss how to automatically create the transaction mechanism, the issues of which transactional properties need to be supported and how the developer specifies which BPMN pattern constitutes a collaborative transaction needs to be addressed – issues that are discussed in the next two subsections 3.2.1 and 3.2.2, respectively. The sub-section 3.2.3 then describes how to create the transactional mechanism in an automated fashion.

3.2.1 Transactional Properties for Collaborative Transactions

To determine the transactional properties, we draw upon previous concepts of transactions in DBs and in blockchains.

DB Transaction Properties

As reviewed in the first section, a relational DB provides a mechanism that allows an application to issue the commands to begin and end a transaction. Any DB reads and write operations, wherein a write represents the ***C***reate, ***R***ead, ***U***pdate, ***D***elete (***CRUD***) operations, between the two start and end transaction commands constitute a transaction. Furthermore, a DB transactional mechanism enforces the ACID properties for a transaction.

Native Blockchain Transactions and Their Properties

We use the term "native blockchain transaction" to refer to the usual definition of a blockchain transaction as a result of the ledger reads and writes made by an execution of any one of the smart contract method. Blockchains support the ACID properties for a native blockchain transaction: Ledger writes are not applied on the blockchain immediately but are collected into a block. Each transaction within a block, which is to be appended to the chain, is checked for consistency, that is correctness of execution of concurrent transactions, against the existing transactions already recorded on the chain and against the other transactions within the same block. Thus, in DB-speak, the property of serializability, and hence *consistency,* of transactions is assured; or in the blockchain-speak, double spending is prevented. *Atomicity* and *isolation* are also assured as the transaction writes to the ledger are not visible to other transactions until the block is approved and appended to the chain, wherein the action of approving and appending a block to the chain is *atomic*. Finally, *durability* is also supported as the chain is replicated.

Specifying and Supporting Multi-method Blockchain Transactions and Their Properties

Blockchains, however, do not provide any concept of a transaction that is the result of an execution of more than one method, whether it is executions of separate methods or repeated execution of the same method, or some combinations of the two. We refer to such a transaction as a ***m***ulti-***m***ethod (***mm***) transaction in [Liu 2023], where we described how to define an mm-transaction as a subset of the methods of a smart contract such the transactions methods are *independent*. The independence property prevents the transaction methods to make references to objects, or invoke methods, that are declared externally to the methods of the transaction as that would violate the atomicity and isolation properties. If a transaction invokes an external method X, and the transaction is then aborted, the activities of the method X would also have to be aborted/negated and hence the isolation property would be violated.

In addition to describing how a developer identifies an mm-transaction, [Liu 2023] also shows that the *ACID properties* must be supported, and that *access control* and *privacy* should be supported:

- The *access control* property states that only the actors who are the participants to the transactions may invoke the transaction methods.
- The *privacy* property states that the transaction ledger updates should be visible only to the participants of the transaction while the transaction is in progress. It should be noted that in [Liu 2023], several transaction mechanisms were explored for the blockchain mm-transactions. They all support the ACID and accesses control properties but provide for different levels of privacy protection. The mechanisms were compared in relative terms of effort required by an attacker to subvert the mechanisms, wherein the "relativity" is in terms of whether one mechanism requires more or less effort on the attacker's part in subverting the privacy mechanism than the attacker requires in subverting another privacy mechanism. The costs of the mechanisms were also evaluated.

Transactional properties for BPMN Collaborative Transactions

We adopt the blockchain mm-transaction properties as the transactional properties to be enforced by the BPMN collaborative transaction mechanism:

- *ACID* properties must be supported.
- *Access control* and *privacy* should be supported.

3.2.2 BPMN Collaborative Transaction: How to Specify

BPMN modeling specification [Business Process Model and Notation (BPMN), Version 2.0. 2023] provides the concept of an all-or-nothing transaction only for the subprocess element, which is a BPMN sub-model with one start and one end element. However, as the sub-process is executed within the context of a single actor, it cannot be used to represent the collaborative activities of more than one actor. Here we address the issues of how to determine which BPMN patterns are suitable for representing a collaborative transaction and how the developer indicates to the system which BPMN pattern is to be treated by the system as a collaborative transaction.

Clearly, selecting any subset of BPMN elements from a BPMN model is not appropriate as the pattern selected as a transaction must have the all-or-nothing property. Consider the sample use case, shown in Fig. 1, and the activities of the *Middleman* actor, which are: *Middleman receive order, forward order,* and *order transport.* It might appear that the three activities might be a suitable pattern to be declared as a BPMN transaction; however, difficulties arise with the atomic commitment. If the first two activities are successfully performed but the third one fails, then the transaction is aborted, and the question arises: What to do about the effects of the *forward order* activity that has already been executed, which means that the order was already sent to the *Supplier* actor? The *Supplier* actor might have *produced* the product that is now prepared for transport – and these activities would also have to be aborted and further cascaded aborts might be required. The problem arises because the activities selected as a transaction were not localized in that they affected other activities not belonging to the transaction. This is similar to the isolation property of the transactions in the DB system: *No activities external to the transaction should view updates made by a transaction that has not been completed.*

In terms of the graph representation, the three activities that were selected as a transaction are such that there is a flow of execution, from the selected BPMN pattern of the three activities, to an activity outside the pattern. To avoid such a situation, nodes that are selected to be the pattern representing a transaction must be such that there is no outgoing edge out of any vertices in the pattern to vertices outside the pattern. The exception is one vertex that may have an outgoing edge(s) corresponding to the outgoing flow of execution in the BPMN model when the transaction is completed. And the above requirements are satisfied when a pattern chosen to be a transaction is a SESE subgraph. Once the flow of computation enters a SESE subgraph via its *entry node*, it remains within that subgraph until the computation exits via the subgraph's *exit node* – we refer to this as the *localization property* of SESE subgraphs and we exploit it, together with additional properties exhibited by the SESE subgraphs described below, in defining the BPMN collaborative transactions and in creating the transactional mechanisms to support the BPMN collaborative transactions.

For the developer to determine which of the BPMN patterns are to be supported as collaborative transactions, the DAG representation of the BPMN model is analyzed in order to find all SESE subgraphs. Once all such subgraphs are identified, they are mapped back to the BPMN model and presented to the developer who determines which of the subgraphs should be transformed into a BPMN collaborative transactions. Once the developer decides which of the SESE subgraphs are to be processed as collaborative transactions, the BPMN model is further processed to create a transactional mechanism to enforce the transactional properties.

Consider the sample use case shown in Fig. 1 that was also used in [Weber 2016, Tran 2018, López-Pintado 2019]. The graph is analyzed to find all SESE subgraphs in the DAG representation of the BPMN model. Fig. 4 shows the SESE subgraphs when mapped backed to the BPMN model shown in Fig. 1. It should be noted that that there are many other SESE subgraphs that exists but are not shown in the Fig. 4, as the figure would become too busy. How SESE subgraphs are further classified and how they are used by the TABS tool will be discussed later.

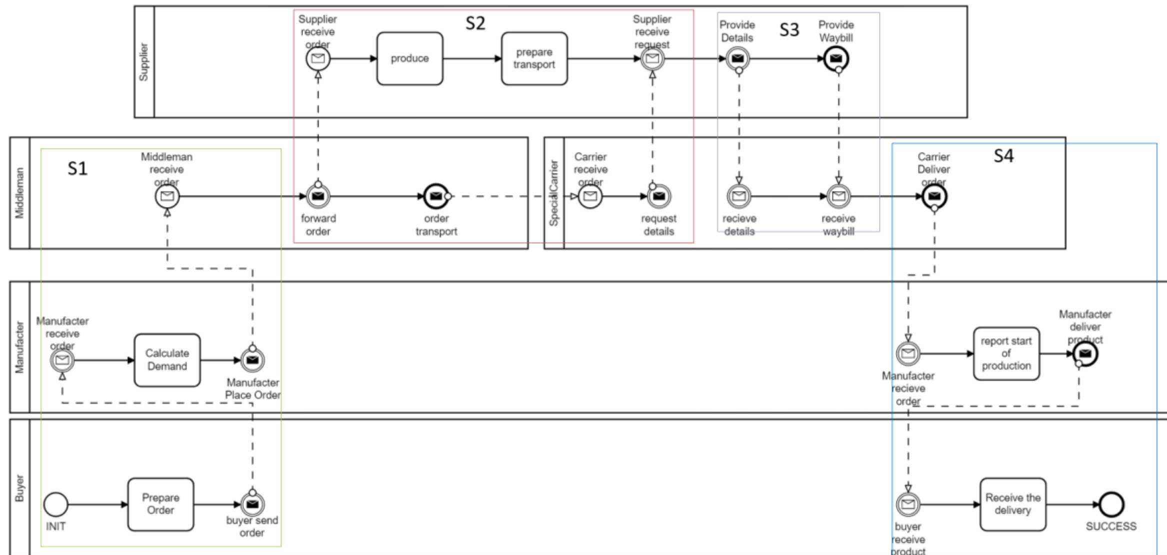


Fig. 4. BPMN model and found subgraphs mapped back to the BPMN model. (Adopted from [Bodorik 2022]).

The subgraphs shown in the figure are:

- S1 … The subgraph contains INIT, Buyer sends offer, Manufacturer receives order, calculate demand, Manufacturer places order and Middleman receives order.
- S2 … The subgraph contains forward order, order transport, supplier receives order, produce, order transport, prepare transport, carrier receive order, supplier receive request and request details.
- S3 … The subgraph contains provide details, provide waybill, receive details, and receive waybill.
- S4 … The subgraph contains carrier deliver order, manufacture receives order, report start of production, manufacturer deliver product, buyer receives product, and SUCCESS.

The developer chooses which of the subgraphs should be used to form a collaborative transaction. In the next subsection we shall discuss the SESE subgraphs property that, if any two SESE subgraphs share vertices, then one of the subgraphs is a proper subset of the other subgraph. This property is used to define nested collaborative transaction that will be discussed in the next section. In this subsection 3.2.2, we described how a BPMN collaborative transaction is identified and specified by the developer, while how a transactional mechanism is created for a BPMN collaborative transaction is described in the next subsection 3.2.3. How the nested collaborative transactions are supported is described in the subsection 3.3.

3.2.3 Transaction Mechanism to Enforce Collaborative Transaction Properties

In this subsection, we describe how the TABS transformations are amended to support the BPMN collaborative transactions. The main effect is on the modeling at the DE-HSM level that is used to determine which BPMN patterns are suitable for defining as collaborative transactions. Thus, BPMN model preprocessing and transformation of the BPMN model into the DAG representation are not affected, and we describe here the effects of supporting the collaborative transactions on DE-HSM modelling and subsequent transformations.

DAG Transformation into DE-HSM and DE-FMS Sub-models and Properties of SESE Subgraphs

As was described above, the developer uses the TABS tool to find and show all SESE subgraphs from which the developer chooses which of the subgraphs are to be treated as collaborative transactions. As was shown in [Bodorik 2022], the SESE subgraphs satisfy the following properties:

- As the BPMN graph representation is a DAG, each of the SESE subgraphs is a DAG.
- As a SESE subgraph is acyclic, the subgraph may be analyzed using multi-modal modeling in which concurrency is modeled using DE modeling and functionality is expressed using HSMs.

There are two consequences due to the above properties:

– In DE modeling, during execution, each subsystem represented by a subgraph contains a queue of events ordered by their timestamps. As none of the SESE subgraphs have feedback loops (they are acyclic), all DE-HSMs models may use just one DE queue of events ordered by the event timestamps.

– More importantly, recall that the edges in the DAG represent the flow of computation. As SESE subgraphs have only one *entry node*, once computation enters the subgraph via the *entry* node, it proceeds with the execution within the subgraph until the execution exists via its *exit node* – hence we refer to a SESE subgraph as representing *localized computation*.

It is the localization property of the SESE subgraphs that we exploited in [Bodorik 2022] for automated generation of smart contracts with sidechain processing, wherein the functionality of a selected SESE subgraph is transformed into the methods of a smart contract that is deployed and executed on a sidechain. It is a "slave" contract of a master smart contract that is executed on the mainchain and that invokes the methods of the slave smart contract. As the slave contract executed on a sidechain has localized computation, once the flow of execution enters the subgraph via its *entry node*, it remains within the subgraph until the flow of computation exits via the subgraphs *exit node*. It should be noted that if a transaction is aborted, then its abort compensation activities are performed within the scope of the transaction and exit from the transaction is via the subgraph's exit node.

There are two additional SESE subgraph properties that were shown in [Bodorik 2022]. For any two SESE subgraphs found to exist in a DAG, they are:

1) Either *mutually exclusive* in that they do not share any vertices, or
2) *one of the subgraphs is a proper subgraph of the other one*.

However, the concept of the SESE subgraphs is further refined to avoid arriving at a model that has too many subgraphs. Consider the graph G of Fig. 5, wherein G = (S, E), S = { s_{en}, v_{11}, v_{12}, v_{21}, v_{31}, v_{32}, v_{33}, s_{ex} }, and E = { (s_{en}, v_{11}), (s_{en}, v_{11}), (v_{11}, v_{12}), (v_{12}, s_{ex}), (s_{en}, v_{21}), (v_{21}, s_{ex}), (s_{en}, v_{31}), (v_{31}, v_{32}), (v_{32}, v_{33}), (v_{33}, s_{ex}) }. G has two independent subgraphs G_1 = (S_1, E_1) and G_2 = (S_2, E_2), where:

i. S_1 = { s_{en}, v_{11}, v_{12}, v_{21}, s_{ex}} and E_1 = { (s_{en}, v_{11}), (v_{11}, v_{12}), (v_{12}, s_{ex}), (s_{en}, v_{21}), (v_{21}, s_{ex}) }
ii. S_2 = { s_{en}, v_{31}, v_{32}, v_{33}, s_{ex}} and E_2 = { (s_{en}, v_{31}), (v_{31}, v_{32}), (v_{32}, v_{33}), (v_{33}, s_{ex}) }

Clearly, it is desirable to have only one DE-HSM model representing the subgraph G as opposed to the two sub-models, one for G1 and one for G2. Further decomposition of G into G1 and G2 is not only unnecessary, but it would also result in

more complex interconnection of the sub-models then is necessary. Another example is an independent subgraph G' = (S', E'), where S' = { sen, v31} and E' = { (sn, v31)} and an independent subgraph G'' = (S'', E''), where S'' = {v32, v33, sex } and E''={ (v32, v33), (v33, sex}). They are both mutually exclusive SESE subgraphs, but clearly, there is no need to have separate sub-models to represent them – hence, in [Bodorik 2022], the concept of Largest Smallest Intendent (LSI) subgraphs was introduced:

LSI subgraph: An LSI subgraph G = (S, I) is a SESE subgraph, which has an *entry vertex* s_{en} and an *exit vertex* s_{ex}, such that for any vertex s ∈ S, which is neither an *entry* nor an *exit vertex* of the SESE subgraph, has exactly one incoming and one outgoing edge.

We note that an LSI subgraph is such that it does not contain any proper subgraph that is also an LSI SESE subgraph. Furthermore, when we make a reference to a SESE subgraph in this paper, we mean a reference to either an LSI SESE subgraph or a SESE subgraph that contains one or more SESE or LSI subgraphs. See [Bodorik 2022] for the full description of the SESE subgraph properties.

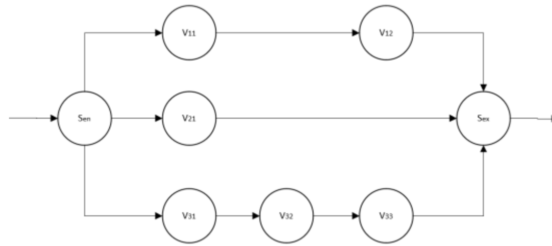


Fig. 5. Independent subgraph with proper subsets that are also independent subgraphs (Adopted from [Elsevier j]).

Transformation of the DE-HSM Sub-models into Smart Contract Methods

Recall from the subsection 2.3 that when there is no sidechain processing, then activities of each actor are represented by a smart contract method representing the flow of execution for that actor. The flow of execution for one actor's model is represented using an FSM, if there is one flow of execution, or using concurrent FSMs if there are concurrent executions within the actor context due to the BPMN flow for that actor having fork gates. However, if the developer chooses a transaction representing a collaboration of several actors, i.e., if the developer chooses a SESE subgraph as a collaborative transaction, our approach represents such a transaction by using a separate flow of execution. Thus, for each SESE subgraph chosen by the developer to be treated as a transaction, a separate smart contract method is created. The flow of execution within that method is represented by the concurrent FSMs that represent the flow of execution within that SESE subgraph chosen by the developer as a transaction. Thus, the smart contract is represented by (n+m) smart contract methods, where n is the number of actors and m is the number of BPMN collaborative unnested transactions as chosen by the developer.

A smart contract method representing a collaborative transaction does not invoke any of the other smart contract methods. Thus, although the smart contract method for a collaborative transaction is invoked multiple times by an application or monitor, the method does satisfy the property that once computation enters the transaction as represented by a SESE subgraph, it stays within that computation until the computation exits the subgraph, which satisfies the *independence property of an mm-transaction* as described in [Liu 2023]. As a consequence, the methods for automated creation of an mm-blockchain transaction mechanisms are applicable also for the transactional mechanism for collaborative transactions as is described next.

Transactional Mechanism for Blockchain Multi-method Transactions

Blockchains support the concept of a transaction that is the result of an execution of a single method and do not support a transaction that is the result of executing more than one method execution, whether it is a multiple execution of the same

method or the result of executing two different methods of the smart contract. In [Liu 2023], we describe how to define an mm-transaction as a subset of the methods of a smart contract under the constraint that such transactions methods are *independent*. The independence property prevents the transaction methods to make references to objects, or invoke methods, that are declared externally to the transaction methods as that would violate the atomicity and isolation properties of a transaction. Recall that the isolation is violated if a transaction method invokes an external method, as the external method may affect computation that is external to the transaction – and if the transaction is aborted, the activities of the method X would also have to be aborted/negated. As described in [Liu 2023], implications of providing a transactional mechanism to support a blockchain mm-transaction are:

1) Blockchain is immutable: Thus, to support an mm-transaction, optimistic methods need to be used in support of the transaction commitment and the transactional properties. An mm-transaction method must perform ledger writes in some private workspace and only when the transaction is committed are the writes written from the private workspace to the ledger.
2) The private workspace to support optimistic transaction methods is needed to store the reads and writes to the ledger before the transaction commit. Only when the transaction commits are the cached data applied to the ledger. Furthermore, the private workspace must be shared across executions of the transaction methods.

However, blockchains may not provide a data structure that persists across executions of methods of a smart contract. An example of a persistent data structure provided by a blockchain is private data provided by the Hyperledger Fabric blockchain in form of a private blockchain shared by specified actors only. However, if such a data structure is not provided, then there is no alternative but to store the shared information on the blockchain itself, potentially with a support for off-chain storage techniques.

To automatically generate a transaction mechanism, we used the pattern augmentation techniques [Liu 2023]. The developer first creates the smart contract including the methods that are intended to constitute the transaction, wherein the transaction methods must satisfy the independence property. Once the smart contract methods are created, the developer identifies/marks the transaction methods so that they are known to the preprocessor as methods belonging to the mm transaction. Before the smart contract methods are compiled, they are processed by a preprocessor that amends each of the transaction method in the way described below. Besides the smart contract methods, the preprocessor is provided with additional information on who the actors are that participate in the transaction and information on how the private workspace is provided, which is discussed later. In short, the preprocessor amends the smart contract methods with patterns as follows:

- The pattern representing the *begin transaction* method is inserted. It initializes the private workspace that must persist across executions of the transaction methods and be accessible to all methods of the transaction. The private workspace acts as a cache to store:
  - The state of the transaction that is managed only by the patterns augmented by the preprocessor. In other words, the transaction methods prepared by the developer are not aware neither of the cache nor the object that is used to store the state of the transaction – these are augmented by the preprocessor.
  - The cache the ledger reads and writes performed by the transaction methods.
- Each read or write to the ledger made by any of the transaction methods must be replaced with a pattern to make that read or write using the cache instead.
- End of the transaction method is inserted. It propagates all transaction reads and writes, made using the cache, to the ledger itself.

The begin and end transaction methods and the declaration and manipulation of the object representing the state of the transaction are facilitated by the preprocessor using the pattern augmentation techniques.

[Liu 2023] provides the following options for providing the private workspace/cache:

1) Cache is hosted in the private data structure if it is provided by the native blockchain.
2) The cache is hosted on the ledger in locations that are known only to the methods of the smart contract and hence the cache is inaccessible by other non-transaction methods.
3) Slave smart contract is used to host the cache and the methods of the mm-transaction. The methods are invoked by the main smart contract that contains all non-transaction methods that invoke the transaction methods of the slave smart contract. There are two variations:
   a. The slave smart contract is deployed on the same chain as the main smart contract.
   b. The slave smart contract is deployed on a sidechain, while the main smart contract is deployed on the mainchain.

It is shown in [Liu 2023] that all the above options support the ACID properties. How augmentation is used to support the properties of access control and privacy is also described. Access control assures that only the actors participating in the transaction can invoke the transaction method. This is attained by augmenting each transaction method with a pattern that checks that the actor, causing the invocation of the transaction method, is in the list of the actors that may participate in the transaction.

There are difficulties with supporting the privacy property that states that only the actors that participate in the transaction have access to the cache. Although access control to the methods is supported, the issue is that any actor that has access to deploy smart contracts on the blockchain has access to any data that is on the blockchain ledger. For privacy, [Liu 2023] examines architectures available to support the transaction as described above. For instance, if the transaction methods are deployed in a separate smart contract on a sidechain, privacy is supported as long as the actors who do not participate in the transaction do not have privileges to access the sidechain. Another option for privacy is to use encryption on the private workspace; however, the cost estimates indicate that this may be expensive if public-key cryptography is used to encrypt all data stored in the private workspace. As an alternative, in order to decrease the cost of cryptography, instead of encrypting all data stored in the cache, which is on the ledger, only the location of the cache on the ledger is encrypted: If an attacker does not know where on the ledger the data is stored, the attacker cannot view the stored data.

Cost estimates for each of the alternative approaches is also provided by comparing the costs. Further details may be found in [Liu 2023].

3.3 Nested BPMN Collaborative Transactions

Consider Fig. 4 that shows the BPMN model of our use with the four SESE subgraphs identified by the TABS tool. As already noted, there are other SESE subgraphs that contain some other LSI or SESE subgraphs. For instance, Fig. 6 shows an additional SESE subgraph, S5, that contains proper subgraphs S1 and S2, wherein each of S1 and S2 is an LSI SESE subgraph as it does not contain any proper subgraphs that are also SESE subgraphs. In terms of selecting the collaborative transaction, there are several options available to the developer, such as:

1. Select S1 and/or S2 as separate collaborative transactions.
2. Select S5 as a collaborative transaction (but without selecting S1 or S2 as a sub-transaction).
3. Select S5 as a collaborative transaction and select one or both of S1 and S2 as sub-transactions.

The last option results in nested transactions, which is an issue that we address in this sub-section. We shall concentrate on describing a simple 2-level nesting of a transaction with two sub-transaction and then we comment on how the approach is generalized to an arbitrary number of nesting levels and sub-transactions.

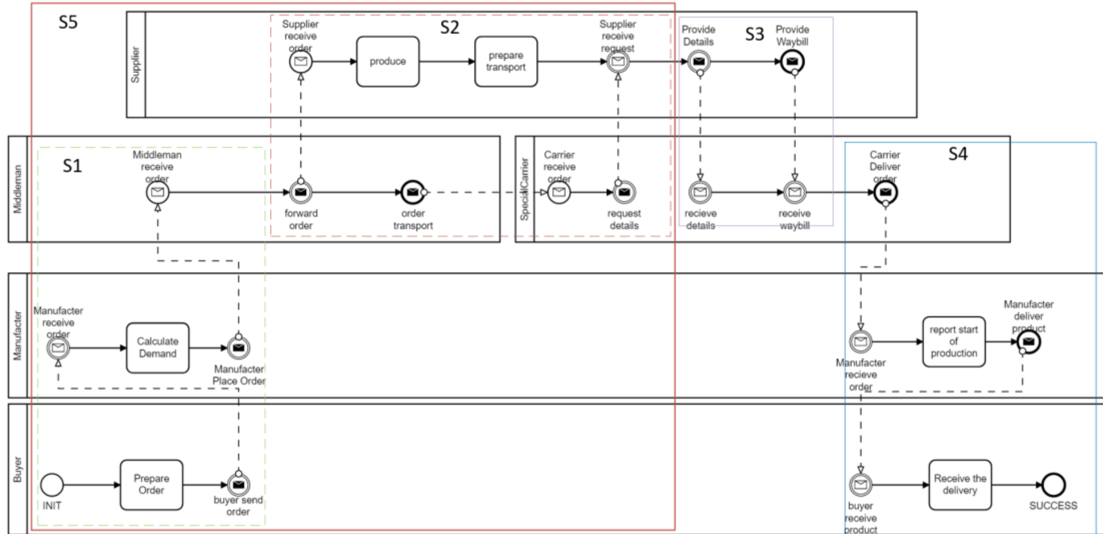


Fig. 6. SESE subgraph S5 contains proper LSI SESE subgraphs S1 and S2.

If the user selects only S1 and S2 as two separate collaborative transactions, then the smart contract will have (n+2) smart contract methods: One method for each of the n actors of the contract, plus one method for each of S1 and S2. For each of the collaborative transactions S1 and S2, pattern augmentation techniques described in the previous section are used to amend the smart contract methods with patterns to create a transactional mechanism for each of S1 and S2.

When a developer selects a transaction with nested sub-transaction, the construction of the transactional mechanism for the nested transaction proceeds bottoms up, starting with creating the mechanism for the innermost transaction. First, the smart contract methods for the innermost transactions are constructed in the usual manner, one method for each of the innermost collaborative transactions as described above. If there is a collaborative transaction with m transactions nested within it, then there will be (n+m+1) smart contract methods: One for each of the actors of the smart contract, one for the outer transaction (the parent) transaction, and one smart contract method for each of the child sub-transactions.

However, atomic commitment needs to be supported for the parent and its child sub-transactions; that is, the commitment of the parent and child transitions together must be atomic. To support the atomicity, we use the 2-Phase-Commit (2PC) protocol, wherein the parent-transaction smart-contract method is augmented with a pattern for the 2PC coordinator of the 2PC protocol, while each of the sub-transactions is augmented with the pattern for the 2PC participant. Thus, before the parent transaction may commit, each of its participating sub-transactions needs to be prepared to commit. Once the parent and children of the 2PC are ready to commit, the parent commits and then child sub-transactions commit.

Recursive application of the above approach yields transactional support for nested transaction to an arbitrary level. As the 2PC protocol is a standard technique that has been used extensively in distributed systems, we shall not elaborate on details, but rather estimate the overhead caused by supporting nested transactions in the next section.

4 TABS+: PROOF OF CONCEPT AND COSTS

As a PoC that our approach to supporting nested BPMN transactions is feasible, we amended the previously described TABS prototype tool to also support the collaborative transactions with nesting and thus create a tool called TABS+. We first show how the tool is used to support a BPMN collaborative transaction without nesting and provide cost estimation. We then show in the subsection 4.2 how the nested transactions are supported and provide cost estimation for the support of the nested transactions.

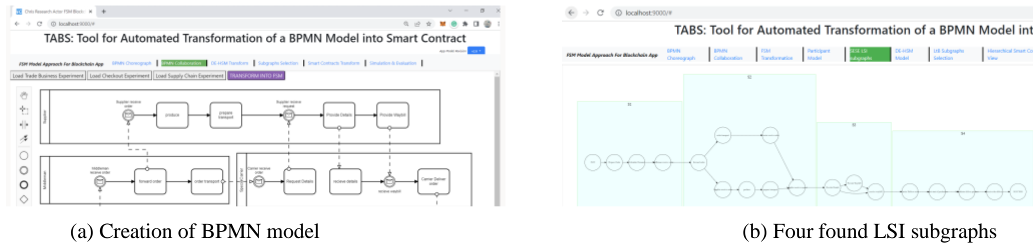
4.1 Collaborative Transactions without Nesting

Fig. 7 shows several screenshots for the use case shown in Fig. 1 and shown in Fig. 4:

– Fig. 7(a) shows the BPMN model that was created using the TABS tool (TABS simply invokes the Camunda BPMN software editor [Camunda 2023] to support the creation of a BPMN model and store it in as an XLM file [Business Process Execution Language 2023].

– Fig. 7(b) shows the LSI SESE subgraphs forming the DE-FSM models, i.e., when each subgraph does not have any proper subgraph that is an LSI SESE subgraph itself, as generated by the TABS+ tool.

1.



(a) Creation of BPMN model (b) Four found LSI subgraphs

Fig. 7. TABS+ tool use: Creation of a BPMN model finding LSI SESE subgraphs

Recall that the BPMN model goes through a series of transformations resulting in the DAG representation as an interconnection of the LSI subgraphs, wherein each LSI subgraphs is represented as a DE-FSM model that may include concurrent FSMs to represent the concurrent streams of executions within a sub-model represented by an LSI subgraph. The interconnection of the LSI subgraphs is shown in Fig. 7(b).

Fig. 8 is a partial screen of the tool showing the SESE subgraphs for selection to be treated as collaborative transactions. All SESE subgraphs are listed, while the first four subgraphs, S1, S2, S3, and S4, are the four LSI subgraphs shown in Fig. 7(b). The user selects in the right-hand-side pane which of the 10 subgraphs, S1, S2, …, S10, are to be treated as collaborative transactions.

In the subsection 3.2.3, we described three architectural alternatives for the collaborative transaction mechanism, alternatives that depend on the selection of the private workspace to store the transaction's state and the ledger access. We now discuss if and how the TABS+ supports each alternative.

a) *Cache is hosted in the private data structure*, if such a data structure is provided by the native blockchain: We *do not* support this alternative in the TABS+, as it requires specialized treatment, depending on how the private data structure is implemented in the native blockchain. Recall that the private data structure must persist across invocations of the transaction methods and must be sharable by all actors participating in the transaction.

b) *The cache is hosted on the ledger* and all methods are in the same smart contract: As the ledger is accessible to any user who has access to the ledger, an attacker, who has access to the ledger, has access to the scripts and hence it conceivable for the attacker to deduce form the smart contract methods' scripts where the private workspace is

located on the ledger and hence access it. Consequently, the method will provide less privacy than the other methods for supporting the transactional mechanism that are described below.

c) Slave smart contract is used to host the cache and the methods of the collaborative transaction: The methods are invoked by the main smart contract that contains all non-transaction methods. There are two variations:
   i. The slave smart contract is deployed on the same chain as the main smart contract.
   ii. The slave smart contract is deployed on a sidechain, while the main smart contract is deployed on the mainchain.

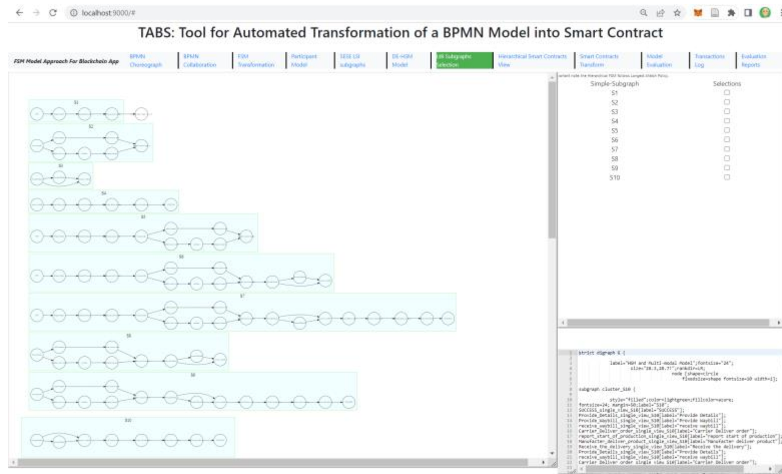


Fig. 8. TABS+ tool: Listing and selecting SESE subgraphs

As we do not support the first case and the last case had two subcases, the TABS+ tool supports the following three alternatives to achieve the support for BPMN collaborative transactions:

1. sc-all… One smart contract is used to host all methods, methods or non-transaction methods.
2. sc-2m … Two smart contracts, one is used to host the transaction methods and the cache, while the second one is used to host all non-transaction methods. Both contracts are deployed on one chain, referred to as the mainchain.
3. sc-2s … Two smart contracts, one is used to host all non-transaction methods and is deployed on the mainchain, while the second one, deployed on a sidechain, is used to host all transaction methods.

For the last case, when there are two smart contracts and sidechain processing is used, we used the same cost calculations for both the mainchain and the sidechain. Thus, the estimated cost when there is sidechain processing is higher than when both smart contracts are on the mainchain. As sidechain processing incurs overhead relative to processing on the mainchain only, it is cost-wise advantageous only when sidechain processing is cheap that its cost benefits outweigh the overhead costs associated with sidechain processing.

However, as we are interested in determining the costs for all the above alternatives, we configured the tool to provide smart contracts for all three options. To determine the cost of executing patterns for each of the alternative mechanisms, we utilize the Ethereum blockchain tools that enable estimation of the cost of execution of smart contract methods written in the Solidity language. We use the Remix compiler that provides for compilation of Solidity smart contract method into executable code and provides estimates of the total cost of executing a smart contract method by relatively detailed cost calculation for each of the method's instruction. We use the Remix compiler to measure the cost of execution for each

method of the smart contract for each of the alternative transaction mechanism approaches and for the base case (labeled as *no-xa*) that does not support collaborative transactions. To calculate the cost of the mechanisms, we performed measurements using a smart contract that contains the following methods:

- Smart contract *method m (inputParameter X:objectX)*. It receives a parameter that is an object of a specified size, and it invokes the methods m1 and m2 in that order.
- method *m1(x1: objectX)* … the method writes to the ledger the object passed as an input parameter.
- method *m2(x1: objectX, x2:objectX*) … the method reads from the ledger the object written by the method m1 and then writes it back to the ledger again as a new object.

We use the Remix compiler to measure the gas-cost of execution for each method of the smart contract and thus estimate the relative overhead cost of providing a transaction mechanism when compared to the base case when execution is without a transaction mechanism. Table 1 and Fig. 9 show the cost for each of the alternative mechanism in the numerical and graphical representation, respectively. It's important to mention that we've standardized the gas price at 20 Gwei to maintain consistency with the default gas price of Ganache, which we used for cost evaluation in our preceding research papers [Liu 2022, Bodorik 2022]. The first column of the table contains the label identifying the transaction mechanism. The subsequent columns show the cost estimates for the alternative mechanisms when the collaborative transaction reads and writes a ledger object of size X = 75, 512, 1024, and 1875 Kb. Note that 1875Kb is the maximum block size that can be used, as of writing this document, on Ehtereum for recording one transaction on the ledger [Ethereum's New 1MB Blocksize Limit, 2021].

Table 1: CPU Processing cost estimates (in Gwei)

Label	***7kb***	***512 kb***	***1024 kb***	***1875 kb***
no-xa	4545000	31027200	62054400	113625000
sc-all	9090582	62054982	124109382	227250582
sc-2m	9319500	63621120	124113000	227254200
Sc-2s	9405000	64204800	127242240	232987500

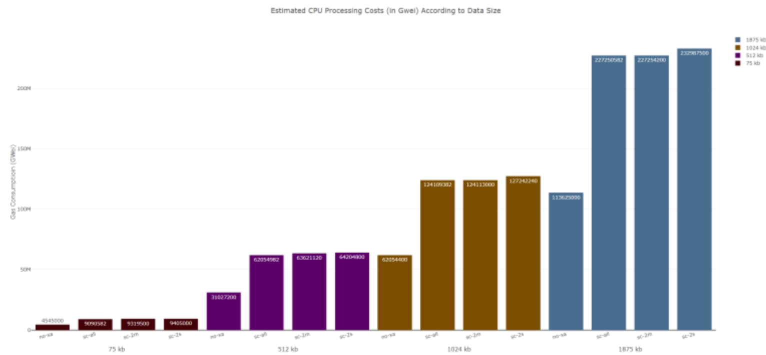


Fig. 9. Estimated CPU processing costs (in Gwei) as a function of data size

The figure and the table show that the cost is indeed directly proportional to the size of the ledger data read/written by each of the approaches. The cost of the transaction mechanism for this particular transaction doubles the cost in comparison to not having transactions. This is not surprising as the data that is read or written is stored in the cache first, and upon the transaction commit the ledger access is replayed from the cache to the ledger itself. Considering that the cache is also on

the ledger, the access to the ledger access in essence doubles and hence the cost, which is directly proportional to the leger reads and writes, also doubles as the table and the figure indicate.

All of the alternative mechanisms we described enforce the ACID properties of transactions as defined in Section 3.2 and also enforce access control. Although each one also supports privacy, the level of privacy protection differs as different levels of efforts are required on the part of the attacker to subvert privacy. We assume that the attacker has access to the blockchain and hence is able to read any object stored on the ledger as long as the attacker knows where on the ledger it is stored – thus privacy may be subverted for the case when all methods and the private workspace are all hosted in the same smart contract (case labelled as *sc-all*).

For the case labelled as *cs-2m*, when the transaction methods and the private workspace are hosted in a separate smart contract hosted on the mainchain, the same approach may be used to subvert the privacy as in the case labelled *sc-all*, except that it would have to be applied twice, once on the main smart contract and once on the smart contract that contains the transaction methods and the private workspace.

The case labelled *sc-2s,* when the transaction methods and the private workspace on hosted in the separate smart contract executed on a sidechain, is similar the case labelled *cs-2m,* except that the attacker also needs access to the sidechain where the smart contract is deployed in addition to having access to the mainchain. Thus, additional information and effort are required in comparison to the previous cases.

To further increase the level of privacy, cryptography may be used to ensure the ledger data written by the transaction is not viewable by the non-transaction actors. To measure the cost of encryption/decryption, the methods m1 and m2, which read and write the ledger, are amended with the public-key cryptography used to encrypt/decrypt any ledger read/write. Thus, the cost of encryption/decryption is expected to be directly proportional to the size of the data that is encrypted/decrypted and thus being expensive, as is the case for the code written in the Solidity language for the EVM [Ethereum Virtual Machine (EVM). 2023]. Table 2 and Fig 10 show the cost of cryptography to encrypt/decrypt the written/read ledger data that has the size X = 75, 512, 1024, and 1875 Kb.

Table 2: CPU processing cost estimates (in Gwei)

Label	***7kb***	***512 kb***	***1024 kb***	***1875 kb***
no-xa	4545000	31027200	62054400	113625000
sc-2s	9405000	64204800	127242240	232987500
sc-2s-crypto	18268952	124706420	249411188	456684152

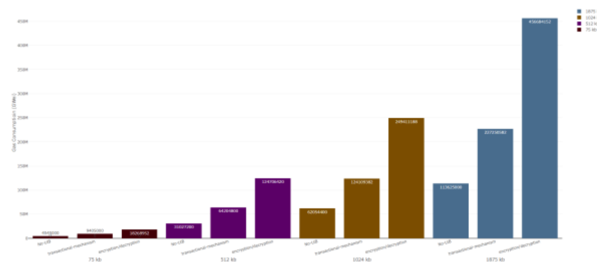


Fig. 10. *Estimated CPU processing costs (in Gwei) for cases labeled as "no-xa", "sc-2s",* and *"sc-2s-crypto"*

The table and the figure show the cost of the work performed when: (i) there is no transaction mechanism, labeled as "no-xa" in the first column of the table; (ii) the transactional mechanism is supported using sidechain processing for the sub-transactions, labeled as "sc-2s"; and (iii) the transactional mechanism is supported using sidechain processing for the

sub-transactions and when the data stored in the private workspace/cache is encrypted/decrypted when writing or reading, labeled as *"sc-2s-crypto"*. Clearly, the cost of encryption/decryption is significant as it doubles the cost of the transactional mechanism when sidechain processing is used (case labeled *"sc-2s-crypto"*).

4.2 Supporting Collaborative Transactions with Nesting

Recall that steps performed in transformations of a BPMN model leading to and finding the SESE subgraphs are the same, whether the tool is used to support sidechain processing or for supporting BPMN collaborative transactions. However, once the SESE subgraphs are shown to the user, as in Fig. 8, further transformation and processing to support the collaborative transaction differs from the case when the TABS tool is used to support sidechain processing. To support a collaborative transaction, the developer is asked to select those SESE subgraphs that should be treated as a BPMN collaborative transaction. For instance, the developer may select the subgraphs S3, S4 and S5 as collaborative transactions. S3 and S4 are LSI subgraphs and hence do not contain other SESE subgraphs. However, the subgraph S5 contains the LSI subgraphs S1 and S2 as its subgraphs and hence they are shown to the user, as is shown in Fig. 11. They may lead to nested transactions in that S5 contains sub-transactions S1 and S2. The developer may then choose any one of S1 and S2, or both, or neither as sub-transactions. Following this, the developer is provided with options for the selection of the type of the collaborative transaction mechanism to be used to support the selected collaborative nested transactions.

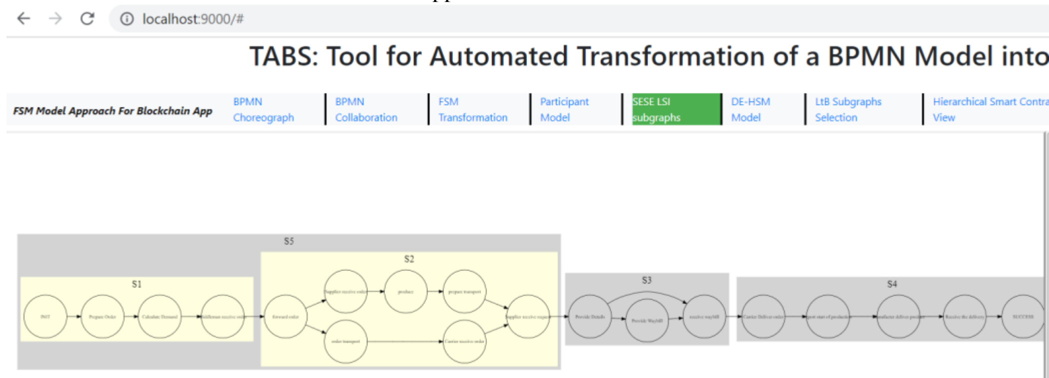


Fig. 11. User selecting subgraphs S3, S4, and S5, wherein S5 contins LSI subgraphs S1 and S2.

The graphical representation of the methods for the collaborative smart contracts are shown in Fig. 12. For each of the collaborative transactions S3, S4, and S5, there is a method that is denoted by three icons labelled "method_start_SX", "tx_SX", and "method_end_SX", where X is either 3, 4, or 5 depending on for which transaction it is. However, as S5 has two sub-transactions S1 and S2, it further contains two methods, one each for the sub-transactions S1 and S2, respectively. The figure's layout was produced using Graphviz [Graphviz 2023], while the visual styling was implemented with D3.js [Bostock 2023]; both Graphviz and D3.js function as libraries within the Node.js framework [Node.js 2023].

Recall that to support the atomicity of the parent transaction with its sub-transactions, a 2PC protocol is used, with each parent transaction acting as a coordinator of the 2PC protocol, while each child sub-transaction acts as a 2-PC participant. Thus, S5 includes the functionality of a 2PC coordinator for its sub-transaction S1 and S2, while each of S1 and S2 contain the functionality for the 2PC of a participant. It should be noted that, as a transaction may be a parent transaction and a child transaction at the same time, it may include the functionality for acting as a coordinator for its sub-transactions and as a participant when it is a sub-transaction within a parent transaction.

The cost estimates for a transaction mechanism to support sub-transactions are expected to be high. In comparison to the cost of a transaction mechanism for a collaborative transaction that is not nested, for each nested transaction a 2PC is implemented. The cost is incurred by a parent transaction coordinator exchanging the 2PC protocol messages with its participants. Figure 13 shows the total cost estimates for performing the 2-PC protocol that consists of the coordinator issuing the Prepare-to-commit command and receiving a response from each of the participants, and then issuing the Commit-command to each of the participants. It also includes the cost for each of the participants to receive the Prepare-to-commit command and to send the Ready-to-commit responses to the coordinator and then receiving the Commit-command from the coordinator (the participant does not acknowledge the received Commit-command response). The cost of the 2PC-commit protocol coordination needs to be added to the rest of the cost of the transactional mechanism.

To estimate the cost of the 2PC protocol implementation as a part of the mechanism to support the nested transaction, we thoroughly evaluated the cost of each step of the protocol, aligning it with the workflow displayed as a sequence diagram in Figure 13.

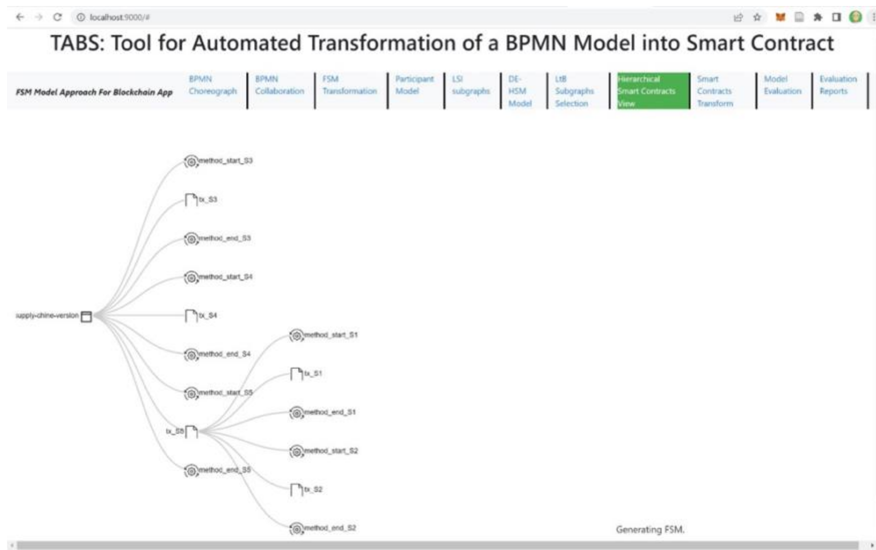


Fig. 12. Methods for collaborative transaction S4, S4, and S5, whrein S5 has sub-trnasactions S1 and S2.

Recall that to support the atomicity of the parent transaction with its sub-transactions, a 2PC protocol is used, with each parent transaction acting as a coordinator of the 2PC protocol, while each child sub-transaction acts as a 2-PC participant. Thus, S5 includes the functionality of a 2PC coordinator for its sub-transaction S1 and S2, while each of S1 and S2 contain the functionality for the 2PC of a participant. It should be noted that, as a transaction may be a parent transaction and a child transaction at the same time, it may include the functionality for acting as a coordinator for its sub-transactions and as a participant when it is a sub-transaction within a parent transaction.

The cost estimates for a transaction mechanism to support sub-transactions are expected to be high. In comparison to the cost of a transaction mechanism for a collaborative transaction that is not nested, for each nested transaction a 2PC is implemented. The cost is incurred by a parent transaction coordinator exchanging the 2PC protocol messages with its participants. Figure 13 shows the total cost estimates for performing the 2-PC protocol that consists of the coordinator issuing the Prepare-to-commit command and receiving a response from each of the participants, and then issuing the

Commit-command to each of the participants. It also includes the cost for each of the participants to receive the Prepare-to-commit command and to send the Ready-to-commit responses to the coordinator and then receiving the Commit-command from the coordinator (the participant does not acknowledge the received Commit-command response). The cost of the 2PC-commit protocol coordination needs to be added to the rest of the cost of the transactional mechanism.

To estimate the cost of the 2PC protocol implementation as a part of the mechanism to support the nested transaction, we thoroughly evaluated the cost of each step of the protocol, aligning it with the workflow displayed as a sequence diagram in Figure 13.

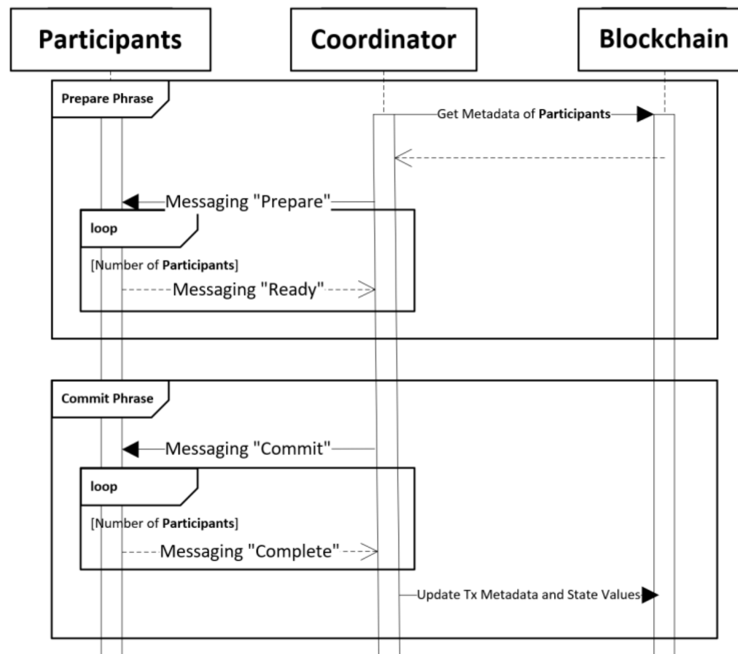


Fig. 13. Sequence Diagram Illustrating the 2-Phase Commit Protocol for Nested Transactions.

During the commitment stage of nested transactions within the parent transaction, the coordinator initiates the 2-phase commit by triggering the 'event' of the smart contracts, which sends a 'ready' message to all its 2-PC participants for the subordinate nested transactions. The execution of this event-emitting function incurs an additional cost. Subsequently, the coordinator awaits responses from the participants through its 'listener' interface. Each subordinate responds by sending back a 'yes' or 'no' message, requiring the response-collecting function to be activated 'N' times, where 'N' equals the number of participants. This is where the overhead of the 2-phase commit protocol is most evident. Similarly, during the second phase, the coordinator disseminates a 'commit' message to all participants, who subsequently respond with their replies. The coordinator declares a successful commitment if all participants return the positive “ready-to-commit” response. The additional cost of the second phase commit is roughly equivalent to that of the first phase.

To gauge the cost of our 2-phase commitment protocol, we employed Remix for interacting with the actual functions of deployed smart contracts. As before, the gas price was set at 20 GWei, identical to the default gas price of Ganache. We estimated the costs of both phases when the number of participants involved in the 2-phase commit varies from 1 to 5, with the results presented in Table 3 for each of the phases of the 2-PC protocol. Figure 14 shows the total cost of the 2-PC,

derived as the sum of the costs shown in Table 3 for Phase1 and Phase2, as a function of the number of participants. As expected, the cost is in a form of a linear function of the number of participants. Furthermore, the cost of a mechanism to support sub-transactions is high due to the 2-PC protocol costs. The cost estimates shown in Fig. 14 need to be added to the cost of the transactional mechanism when nesting is not used, and when they are added, then the cost of the mechanism is more than doubled when there are two nested sub-transactions and increases linearly with an increase to the number of participants.

Table 3: Costs Estimates for the 2-Phase Commit Protocol (in Gwei)

# of participants	*Phase 1*	*Phase 2*
2	627820	627400
3	671500	671080
4	715180	714760
5	758860	758440
6	802540	802120

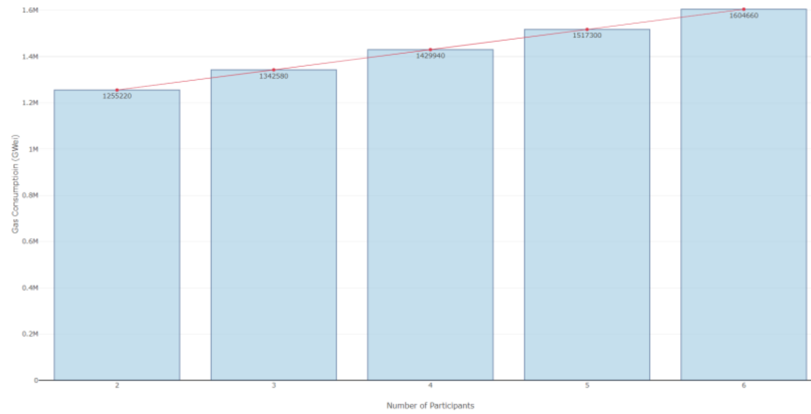


Fig. 14. Gas Utilization by the 2-Phase Commit Protocol for Nested Transactions.

5 CONCLUDING REMARKS AND FUTURE WORK

5.1 Concluding Remarks

This paper described how the TABS approach, to automated creation of smart contracts from BPMN models, is extended to provide automated support for nested collaborative transactions. A collaborative transaction is long term and involves several participants. We showed how a BPMN model can be analyzed to determine which BPMN patterns are suitable to be supported as collaborative transactions that have the desirable ACID, access control, and privacy properties. We also described how a transaction mechanism to enforce the aforementioned properties is created using pattern augmentation techniques. The developer is provided with the information on the BPMN patterns that are suitable for defining as collaborative transactions and which transactional mechanisms are available for their support. The developer chooses which of the shown BPMN patterns are to be treated as the collaborative transactions and which of the transaction mechanism is to be used. The TABS+ approach and tool create the smart contract(s) that contain the collaborative transactions and the supporting transactional mechanism automatically.

Of course, the motivation is to alleviate the developer as much as possible from the responsibilities for creating the code of the smart contract methods. In comparison to other approaches in creating the smart contracts from the BPMN model our approach is unique in that it supports in automated fashion:

- *Sidechain processing*: The developer simply selects which of the patterns, from the list of patterns that are suitable for sidechain processing as provided by the system, should be deployed and executed on a sidechain.
- *Collaborative transactions*: The developer simply selects which of the patterns, from the list of patterns that are suitable to be supported as collaborative transactions, should be deployed and executed as collaborative transactions. Furthermore, the developer has a choice on which of the available transactional mechanism should be used to support the collaborative transactions.
- *Nested collaborative transactions*: The system also supports nesting of the collaborative transactions.

One of the main benefits, however, is that developer's responsibilities are minimized. In essence, the developer need not worry about the which methods should form transactions and need not worry about how to create a transaction mechanism. In fact, our approach avoids the issue of the BPMN not having a way to represent a transaction that is a collaboration of actors as the BPMN model is analysed and the system provides information on which BPMN patterns are suitable as collaborative transactions.

Another underlying benefit due to our approach is that it is semi-agnostic to the blockchain. As long as the monitor part of a smart contract is written for a blockchain, any smart contract developed using our approach is deployable on that blockchain. The reason is that the monitor part of the smart contract is independent of smart contracts because the smart contract is expressed in terms of the DE-HSM model, i.e., in the interconnection of the DE-FSM models which work on any blockchain for which the smart contract monitor is implemented. Of course, there is still the issue of scripting the individual tasks that needs to be performed for a specific blockchain. How that can be avoided is described in the next subsection on future work.

5.2 Future Work

The next extension of this research is to make the approach agnostic to blockchains; i.e., if a smart contract is developed, it should be deployable and executable on any blockchain. We are half-way there in that the collaboration is blockchain independent as it is expressed in terms of interconnection of the DE-FSM models, and it uses DEs to model concurrency and concurrent FSMs to model the functionality. However, currently, to apply a smart contract developed for one blockchain to be deployable and executable on another blockchain, the scripts need to be provided by the developer for the task elements to be executable on the target blockchain. We shall use the standard two-layer approach taken by the Plasma project and others in that the task scripts are not executed on the blockchain, but rather off-chain, while the smart contract simply guides the collaborations and obtains certifications about the results of the tasks that are executed off chain.

Similar to the above, we also plan to have results of each transaction be certified by each of the transaction participants. Instead of executing the tasks as a part of the smart contract, tasks executed by individual participants will be processed off chain, while the results of the computation will be certified by the task executor, while on the blockchain only the results of the task computations are stored together with the certifications of the results.

Another extension is to provide for automated hardening of the smart contract methods by automatically ensuring that the security best practices for the creation of the smart contract methods are followed. We are adapting the approach of [Marvidou 2018] in which a set of security patterns is added to the smart contract methods.

The above extensions to the TABS+ are short-term and are being addressed already. For future longer-term work, we are investigating approaches to augmentation of BPMN models with patterns or approaches to replace certain patterns with

other similar BPMN patterns. The objective is to support a repository of patterns that provide for certain functionality, wherein the patterns are classified and categorized by their functionality and data flow, including inputs to the pattern, outputs from the pattern, and methods used. For instance, a BPMN model can represent collaboration of activities in a letter of credit that was described in the Motivation section 1.1. Recall that the model, in addition to the seller, buyer, and banks, other actors may be involved in order to provide for transport of the product to the port, wherein the transport that may include specific safety requirements and insurance, and to support paper-work for customs when crossing borders, release of the goods from the port, transport to the seller, and other activities. Shipment safety activities and insurance required for the transport of the goods may depend on the type of goods and methods of transport and, furthermore, may also depend on the country or region through which transport crosses. Different BPMN patterns may represent different activities that need to be performed for the transport of different goods using different transports – the repository of patterns is searched for a suitable pattern that, if it exists, can be augmented onto the model instead of having to prepare such pattern from scratch. By supporting automated creation of smart contracts from BPMN models and providing support for augmentation of BPMN models with BPMN patterns and replacement of patterns in BPMN models with similar patterns, we are striving to create an environment to provide a Smart Contract As a Service (SC-as-a-service) that includes the support of automated creation of a smart contract from a BPMN model and the support of its deployment on any blockchain.

6 HISTORY DATES

In case of submissions being prepared for Journals or PACMs, please add history dates after References as (*please note revised date is optional*):

Received June 2023; revised August 2023; September 2023

REFERENCES

[1] Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. https://www.semanticscholar.org/paper/Bitcoin%3A-A-Peer-to-Peer-Electronic-Cash-System-Nakamoto/4e9ec92a90c5d571d2f1d496f8df01f0a8f38596

[2] Marr, B. (n.d.). A Short History Of Bitcoin And Crypto Currency Everyone Should Read. Forbes. Retrieved May 12, 2023, from https://www.forbes.com/sites/bernardmarr/2017/12/06/a-short-history-of-bitcoin-and-crypto-currency-everyone-should-read/

[3] Szabo, N. (2018). Smart Contracts: Building Blocks for Digital Markets. https://www.semanticscholar.org/paper/Smart-Contracts-%3A-Building-Blocks-for-Digital-Szabo/9b6cd3fe0bf5455dd44ea31422d015b003b5568f

[4] Buterin, V. (2015). A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM. https://www.semanticscholar.org/paper/A-NEXT-GENERATION-SMART-CONTRACT-%26-DECENTRALIZED-Buterin/0dbb8a54ca5066b82fa086bbf5db4c54b947719a

[5] Taylor, P. J., Dargahi, T., Dehghantanha, A., Parizi, R. M., & Choo, K.-K. R. (2020). A systematic literature review of blockchain cyber security. Digital Communications and Networks, 6(2), 147–156. https://doi.org/10.1016/j.dcan.2019.01.005

[6] Khan, S. N., Loukil, F., Ghedira-Guegan, C., Benkhelifa, E., & Bani-Hani, A. (2021). Blockchain smart contracts: Applications, challenges, and future trends. Peer-to-Peer Networking and Applications, 14(5), 2901–2925. https://doi.org/10.1007/s12083-021-01127-0

[7] Vacca, A., Di Sorbo, A., Visaggio, C. A., & Canfora, G. (2021). A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges. Journal of Systems and Software, 174, 110891. https://doi.org/10.1016/j.jss.2020.110891

[8] Belchior, R., Vasconcelos, A., Guerreiro, S., & Correia, M. (2021). A Survey on Blockchain Interoperability: Past, Present, and Future Trends (arXiv:2005.14282). arXiv. https://doi.org/10.48550/arXiv.2005.14282

[9] Saito, K., & Yamada, H. (2016). What's So Different about Blockchain? — Blockchain is a Probabilistic State Machine. 2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW), 168–175. https://doi.org/10.1109/ICDCSW.2016.28

[10] Garcia-Garcia, J. A., Sánchez-Gómez, N., Lizcano, D., Escalona, M. J., & Wojdyński, T. (2020). Using Blockchain to Improve Collaborative Business Process Management: Systematic Literature Review. IEEE Access, 8, 142312–142336. https://doi.org/10.1109/ACCESS.2020.3013911

[11] Lauster, C., Klinger, P., Schwab, N., & Bodendorf, F. (2020). Literature Review Linking Blockchain and Business Process Management (pp. 1802–1817). https://doi.org/10.30844/wi_2020_r10-klinger

[12] Levasseur, O., Iqbal, M., & Matulevičius, R. (n.d.). Survey of Model-Driven Engineering Techniques for Blockchain-Based Applications.

[13] Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., & Mendling, J. (2016). Untrusted Business Process Monitoring and Execution Using Blockchain. In M. La Rosa, P. Loos, & O. Pastor (Eds.), Business Process Management (pp. 329–347). Springer International Publishing.

https://doi.org/10.1007/978-3-319-45348-4_19

[14] López-Pintado, O., Dumas, M., García-Bañuelos, L., & Weber, I. (2019). Dynamic Role Binding in Blockchain-Based Collaborative Business Processes. In P. Giorgini & B. Weber (Eds.), Advanced Information Systems Engineering (pp. 399–414). Springer International Publishing. https://doi.org/10.1007/978-3-030-21290-2_25

[15] Tran, A. B., Lu, Q., & Weber, I. (n.d.). Lorikeet: A Model-Driven Engineering Tool for Blockchain-Based Business Process Execution and Asset Management.

[16] López-Pintado, O., García-Bañuelos, L., Dumas, M., Weber, I., & Ponomarev, A. (2019). CATERPILLAR: A Business Process Execution Engine on the Ethereum Blockchain (arXiv:1808.03517). arXiv. https://doi.org/10.48550/arXiv.1808.03517

[17] Bodorik, P., Liu, C. G., & B Jutla, D. (2022). TABS: Transforming automatically BPMN models into blockchain smart contracts. Journal of Blockchain: Research and Applications, 100115. https://doi.org/10.1016/j.bcra.2022.100115

[18] Liu, C., Bodorik, P., & Jutla, D. (2021a). From BPMN to smart contracts on blockchains: Transforming BPMN to DE-HSM multi-modal model. 2021 International Conference on Engineering and Emerging Technologies (ICEET), 1–7. https://doi.org/10.1109/ICEET53442.2021.9659771

[19] Liu, C., Bodorik, P., & Julta, D. (2023). Nested Blockchain Transaction for Multiple Methods of a Smart Contract. Under review. Distributed Ledger Technologies: Research and Practice.

[20] Mendling, J., Weber, I., Aalst, W. V. D., Brocke, J. V., Cabanillas, C., Daniel, F., Debois, S., Ciccio, C. D., Dumas, M., Dustdar, S., Gal, A., García-Bañuelos, L., Governatori, G., Hull, R., Rosa, M. L., Leopold, H., Leymann, F., Recker, J., Reichert, M., … Zhu, L. (2018). Blockchains for Business Process Management—Challenges and Opportunities. ACM Transactions on Management Information Systems, 9(1), 1–16. https://doi.org/10.1145/3183367

[21] Loukil, F., Boukadi, K., Abed, M., & Ghedira-Guegan, C. (2021). Decentralized collaborative business process execution using blockchain. World Wide Web, 24(5), 1645–1663. https://doi.org/10.1007/s11280-021-00901-7

[22] Bodorik, P., G. Liu, C., & Julta, D. (2021). Using FSMs to Find Patterns for Off-Chain Computing: Finding Patterns for Off-Chain Computing with FSMs. 2021 The 3rd International Conference on Blockchain Technology, 28–34. https://doi.org/10.1145/3460537.3460565

[23] Liu, C., Bodorik, P., & Jutla, D. (2021b). A Tool for Moving Blockchain Computations Off-Chain. Proceedings of the 3rd ACM International Symposium on Blockchain and Secure Critical Infrastructure, 103–109. https://doi.org/10.1145/3457337.3457848

[24] Liu, C. G., Bodorik, P., & Jutla, D. (2021c). Automating Smart Contract Generation on Blockchains Using Multi-modal Modeling. 10.12720/Jait, 13(3), 213–223. https://doi.org/10.12720/jait.13.3.213-223

[25] Liu, C. G., Bodorik, P., & Jutla, D. (2022). Supporting Long-term Transactions in Smart Contracts. 2022 Fourth International Conference on Blockchain Computing and Applications (BCCA), 11–19. https://doi.org/10.1109/BCCA55292.2022.9922193

[26] Business Process Model and Notation (BPMN), Version 2.0. (n.d.). Retrieved from https://www.omg.org/spec/BPMN/2.0.2/PDF (access 12 May 2023).

[27] Dikmans, L., 2008. Transforming BPMN into BPEL: Why and How. https://www.oracle.com/technical-resources/articles/dikmans-bpm.html (access 26 March 2022).

[28] Deehan, N. (2021, November 26). How Developing Java Apps is Easier with a Process Engine. Camunda. https://camunda.com/blog/2021/11/how-developing-java-apps-is-easier-with-a-process-engine/

[29] Harel, D. (1987). Statecharts: A visual formalism for complex systems. Science of Computer Programming, 8(3), 231–274. https://doi.org/10.1016/0167-6423(87)90035-9

[30] Girault, A., Lee, B., & Lee, E. A. (1999). Hierarchical finite state machines with multiple concurrency models. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 18(6), 742–760. https://doi.org/10.1109/43.766725

[31] Yannakakis, M. (2000). Hierarchical State Machines. Proceedings of the International Conference IFIP on Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics, 315–330.

[32] Hoare, C. A. R. (1978). Communicating sequential processes. Communications of the ACM, 21(8), 666–677. https://doi.org/10.1145/359576.359585

[33] Cassandras, C. (1993). Discrete event systems: Modeling and performance analysis. https://www.semanticscholar.org/paper/Discrete-event-systems-%3A-modeling-and-performance-Cassandras/0e132ecc5400d9bcd9e45f482192a3f66de13475

[34] Steichen, M., Fiz Pontiveros, B., Norvill, R., Shbair, W., & State, R. (2018). Blockchain-Based, Decentralized Access Control for IPFS. https://doi.org/10.1109/Cybermatics_2018.2018.00253

[35] OBE, P. B. B. (2022, April 13). Building For a Distributed and Resilient Future … Here's Pigpen, IPFS and Ethereum. ASecuritySite: When Bob Met Alice. https://medium.com/asecuritysite-when-bob-met-alice/building-for-a-distributed-and-resilient-future-heres-pigpen-ipfs-and-ethereum-de05d0dee386

[36] Nabben, K. (2022). Decentralized Technology in Practice: Social and technical resilience in IPFS. 2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW), 66–72. https://doi.org/10.1109/ICDCSW56584.2022.00021

[37] BPMN 2.0 Introduction · Flowable Open Source Documentation. (n.d.). BPMN 2.0 Introduction · Flowable Open Source Documentation; www.flowable.com. Retrieved March 29, 2022, from https://www.flowable.com/open-source/docs/bpmn/ch07a-BPMN-Introduction

[38] Dijkman, R., Dumas, M., & Ouyang, C. (2008). Semantics and analysis of business process models in BPMN. Information and Software Technology, 50, 1281–1294. https://doi.org/10.1016/j.infsof.2008.02.006

[39] Camunda. BPMN 2.0 Symbols—A complete guide with examples. (n.d.). Retrieved May 12, 2023, from https://camunda.com/bpmn/reference/

[40] Business Process Execution Language. (2022). In Wikipedia. https://en.wikipedia.org/w/index.php?title=Business_Process_Execution_Language&oldid=1111618084

[41] Research, B. (2021, December 2). Ethereum's New 1MB Blocksize Limit. BitMEX Blog. https://blog.bitmex.com/ethereums-new-1mb-blocksize-limit/

[42] Ethereum Virtual Machine (EVM). (n.d.). Ethereum.Org. Retrieved May 12, 2023, from https://ethereum.org

[43] Graphviz. (2023). In Wikipedia. https://en.wikipedia.org/w/index.php?title=Graphviz&oldid=1150306368

[44] Bostock, M. (n.d.). D3.js—Data-Driven Documents. Retrieved May 12, 2023, from https://d3js.org/

[45] Node.js. (n.d.). Node.Js. Retrieved May 12, 2023, from https://nodejs.org/en

[46] Mavridou, A., & Laszka, A. (2018). Designing Secure Ethereum Smart Contracts: A Finite State Machine Based Approach. Financial Cryptography. DOI: 10.1007/978-3-662-58387-6_28