Generation of Smart Contracts by Business Analyst with TABS+R Tool

Christian Gang Liu
Faculty of Computer Science
Dalhousie University
Halifax, Canada
Chris.Liu@dal.ca

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

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

Abstract— **This paper addresses the challenge of generating smart contracts from Business Process Management and Notation (BPMN) models, a key area of blockchain research. In our prior work, we introduced TABS, a methodology and tool that automates the generation of smart contracts from BPMN models. This approach abstracts the BPMN flow control, making it independent of the underlying blockchain infrastructure, with only the BPMN task elements requiring coding. In subsequent research, we enhanced our approach by adding support for nested transactions and enabling smart contract repair and upgrades. To empower Business Analysts (BAs) to generate smart contracts without relying on software developers, we incorporated Decision Management Notation (DMN) to represent business rules that define BPMN task functionalities. DMN allows for simple decision-making through the FEEL language and supports references to fields in semi-structured documents flowing through the process. This paper demonstrates how BAs, even without software development expertise, can use our TABS+R tool to automatically generate and deploy smart contracts on blockchain platforms.**

Keywords — Automated Generation of Smart Contracts, Blockchain, BPMN, DMN, Trade of goods and services

I. INTRODUCTION

The publication of the Bitcoin white paper in 2008 and the subsequent launch of the Bitcoin blockchain in 2009 sparked significant interest and research into blockchain technology. This emerging technology has garnered widespread attention from businesses, researchers, and the software industry due to its key attributes, such as trust, immutability, availability, and transparency. However, as with any new technology, blockchain and its associated smart contracts pose a range of challenges, particularly in areas like blockchain infrastructure and smart contract development.

Ongoing research is tackling several critical issues, including blockchain scalability, transaction throughput, and the high costs associated with consensus algorithms. In addition, smart contract development faces unique obstacles, such as limited stack space, the oracle problem, data privacy concerns, support for long-running contracts, and cross-blockchain interoperability. These challenges have been the subject of extensive study, with numerous comprehensive literature reviews available [e.g., 1, 2].

The inherent constraints of blockchain technology complicate the development of smart contracts, as documented in several literature surveys [e.g., 3, 4]. Consequently, developers must be not only be proficient in traditional software development but also possess expertise in smart contract programming for distributed environments, including the use of cryptographic techniques integral to blockchain infrastructure. To address these challenges and simplify smart contract development, research in [5-8], proposes leveraging Business Process Model and Notation (BPMN) models as a foundation for generating smart contracts.

Our approach also uses BPMN to represent business application requirements; however, we take a different route to transform BPMN models into smart contracts. Our method leverages multi-modal modeling to represent the flow of business logic in a blockchain-agnostic manner, providing unique advantages for automated or semi-automated smart contract creation and deployment. As a proof of concept, we developed a tool called Transforming Automatically BPMN model into Smart contracts with Repair Upgrade (TABS+R), which automates the generation of smart contracts from BPMN models.

It should be noted that the BPMN and DMN are standards created by the Object Management Group (OMG) [9]. Both BPMN and DMN are graphical standards that *have been designed to be readily understandable by non-technical people and serve as a bridge that allows collaboration between business stakeholders and IT personnel*. BPMN is used to represent well-defined business processes, while DMN is used to specify precisely business decisions and rules. The DMN standards specifies the use of the Friendly Enough Expression Language (FEEL) that was designed to write expressions in a way that is *easily understood by both business professionals and developers.* FEEL is used to define expressions in the context of BPMN and DMN diagrams [9].

A. Objectives

The main objective of this paper, which also forms its contribution, is to describe how a Business Analyst (BA) uses our TABS+R tool to generate smart contracts from BPMN models, augmented with DMN models used to describe business decision logic.

B. Outline

In the second section, we outline our system architecture for creating smart contracts and for their execution. We overview the significant features of our approach to automated development of smart contracts from BPMN and DMN models. The third section concentrates on interaction of the BA with the system to demonstrate the use of our TABS+R tool in creation and deployment of smart contracts. The final sections provide review of related literature and conclusions.

II. Using DE-FSM Modeling to Write Smart Contract

Instead of transforming the BPMN models directly to smart contract methods, our approach exploits multi-modal modeling to represent the flow of computation of the business logic in a blockchain-agnostic manner [10]. We subsequently extended our approach and methodology resulting in an approach, and a PoC tool called TABS+R [11, 12] to support:

- Automated generation of smart contracts from BPMN models. If the BPMN task elements can be represented using DMN and FEEL, then smart contracts can be generated automatically from BPMN and DMN models prepared by a BA without the support of a developer.
- Support for long running and multi-step transactions.
- Repair/upgrade of smart contracts.

The overall architecture of our system is illustrated in Fig. 1. It presents a block diagram outlining the key steps involved in transforming a BPMN model into smart contract methods. The diagram also includes a set of API methods (denoted as DAppAPI in Fig. 1) that facilitate interaction between a Distributed Application (DApp) and the smart contract methods. This architecture is typical of most approaches that generate smart contract methods from BPMN models [4-6]. In this setup, the DApp does not directly invoke the smart contract methods. Instead, it calls API methods provided by the *API-SCmethods* component in Fig. 1), which marshal the necessary parameters and then trigger the corresponding smart contract methods.

During the design phase, the activities of actors involved in the smart contract are represented using multi-modal modeling. Concurrency is modeled using Discrete Event (DE) modeling, while functionality is represented with concurrent Finite State Machines (FSMs), forming a DE-FSM model. A key feature of this model is its blockchain-agnostic nature, meaning that the coordination of collaborative activities is described using DE modeling. Only the code for the BPMN task elements is blockchain-dependent, i.e., it is generated in a programming language that is supported by the target blockchain. For example, Ethereum-based blockchains typically use languages that produce code executed by the Ethereum Virtual Machine (EVM), whereas JavaScript or other languages may be used for scripting task elements in Hyperledger Fabric (HLF) blockchains.

Scripting task elements is relatively straightforward compared to scripting the collaboration of independent activities. Task code implementing a task is self-contained: Once the flow of computation enters a task element and its execution begins, the task completes its computation without interruption. Furthermore, task code can only access state variables or information flowing into the task, and it produces output information that flows out of the task when its computation finishes.

We use DMN to express business rules within a task element, with DMN leveraging the FEEL language for decision logic. As will be explained later, we also provide automated transformation of the decision logic expressed in DMN and FEEL into code compatible with the native language of the target blockchain. As a result, *when the business logic for the BPMN task elements is defined using DMN and FEEL, our approach enables the automatic generation of smart contracts for the target blockchain from BPMN and DMN models.* This process is fully automated, allowing a Business Analyst (BA) to create and deploy smart contracts without the need for assistance from software developers.

In summary, the flow of collaborative activities is modeled using DE-FSM (Discrete Event - Finite State Machine) modeling, where concurrency is expressed through DE modeling and concurrent FSMs. The functionality of task elements is achieved by invoking methods that implement the business logic of each task. To coordinate these collaborative activities, a run-time monitor, implemented as a smart contract method deployed on the target blockchain, ensures the correct sequencing and execution of the activities. This monitor uses DE modeling to manage the invocation of individual activities, which are represented as methods within the monitor.

For the automated generation of smart contracts, the monitor smart contract must be deployed on the target blockchain. In our proof of concept (PoC), we implemented the monitor smart contract to be deployed on the Hyperledger Fabric (HLF) blockchain as well as on blockchains supporting the EVM [10-12].

III. BPMN Modeling by BA

For explanatory purposes, we will use a simple BPMN model, shown in Fig. 2, that represents a sale of a large product, such as a combine harvester. The model shows that an agreement on the sale of the product is reached first, followed by arrangements for transporting the product. These transport arrangements include determining the requirements for transporting the product, such as safety measures for hazardous materials. Once the transport requirements are established, insurance and transport are arranged, and the product is shipped. After transportation, the product is received, and payment is completed.

As is the usual practice for blockchains, a document file or a large object is not stored on the blockchain itself but is stored off-chain. For storage of documents, we currently utilize the Inter Planetary File System (IPFS) [13] for its reliability and availability supported through replication.

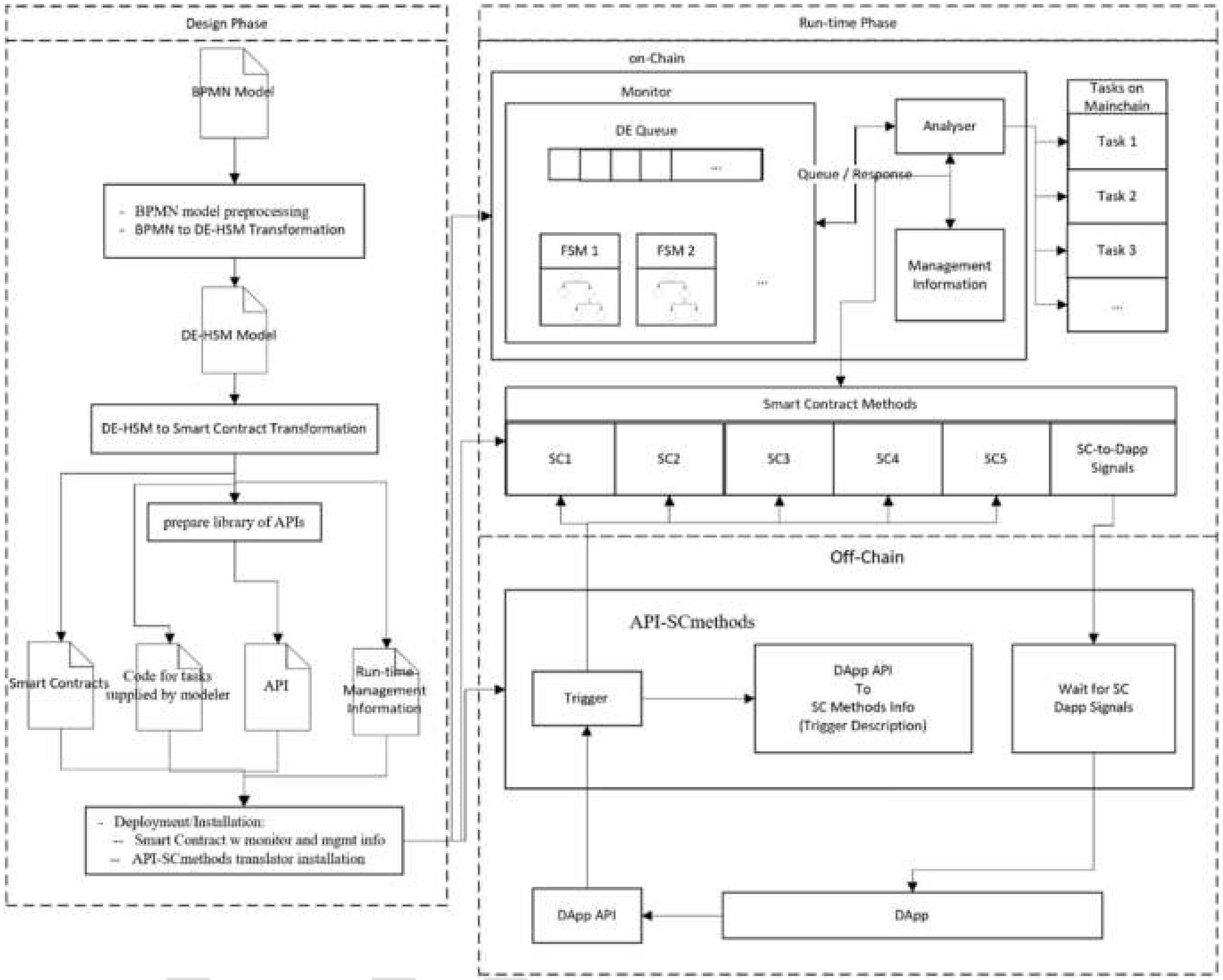

Fig. 1. Sysgtem rhictecture for the design phase and for the execution phase

When a document is created and uploaded to IPFS, a new Content-addressed hash code IDentifier (CID) is generated. This CID is then signed and stored by the smart contract, providing a method to verify the document's authenticity, including confirming (i) authorship and (ii) immutability—ensuring that the document has not been altered since its creation.

Modeling is carried out by a Business Analyst (BA) who is assumed to be proficient in BPMN and DMN modeling, including the use of the FEEL language for decision logic. Additionally, we assume that the BA is familiar with JavaScript Object Notation (JSON), which is used to describe the flow of information throughout the computation process. This will be detailed later.

In Fig. 2, the first task, *RecAgr,* involves receiving a purchase offer document from an external source. Once accepted, the purchase agreement (i.e., a sales agreement) is passed to the next task, *GetTrReq*, for further processing. The sales agreement is represented by an associated data element, *SalesAgr*. The *GetTrReq* task determines the transport requirements for the product and stores them in a newly generated IPFS document, *TrRequirements.* This document is then passed to the subsequent processing step.

The transport requirements are forwarded to the *GetIns* and *GetTransp* tasks to secure insurance and a transporter, respectively. These tasks can be executed concurrently, as indicated by the fork gateway (diamond shape with a plus sign). The *GetIns* task generates the insurance contract, labeled *Insurance*, while the *GetTransp* task produces the *Transport* document.

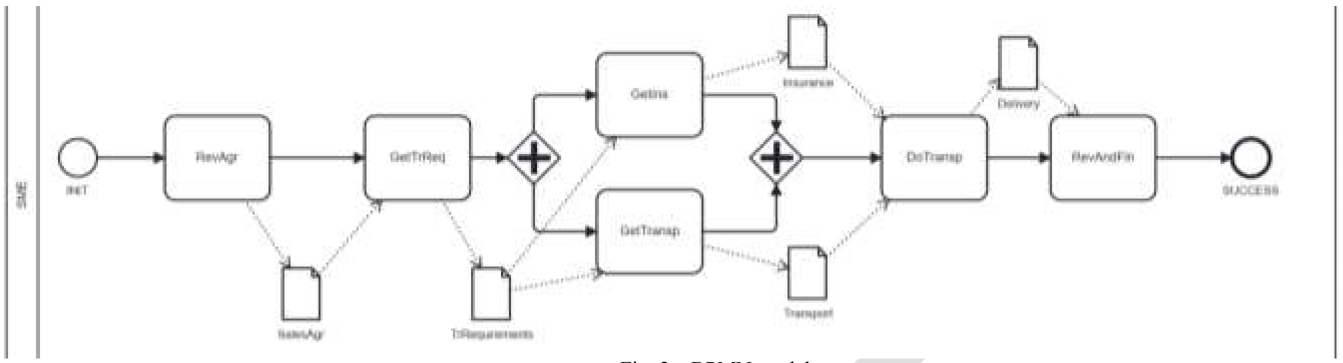

Fig. 2. BPMN model

Once both the insurance and transport contracts are obtained and provided to the transporter, the product is delivered to its destination, represented by the *DoTransp* task. The the product has been received by the purchaser and the contract is completion of the delivery is recorded in a document called *Delivery*, which is then forwarded to the final task, *RecAndFin*, to indicate that considered finalized.

It is important to note that the flow of activities shown in Fig. 2 is executed by a single actor, represented within one swimlane in BPMN terminology. This model is suitable for scenarios within organizations that lack sophisticated IT infrastructure, such as Small to Medium-sized Enterprises (SMEs). The simple use case is designed to demonstrate how the BA uses BPMN elements to document the flow of information along the computation process and how the BA applies DMN modeling to define the business logic.

One of the key features that supports trust in smart contracts is that the methods within a smart contract do not have access to external resources, such as file systems or communication subsystems. Smart contract code can only access the state variables stored on the blockchain and the parameters passed to smart contract methods when invoked. Therefore, any additional information required by a smart contract, beyond the state variables, must be marshaled by the *API-SCmethods* component before the smart contract method is invoked. The marshalled data is then passed as input parameters.

Additionally, a smart contract must communicate the progress of its execution to the Distributed Aplication programs (*DApps*) that invoke its methods. This is accomplished by emitting events from the smart contract methods, which are captured by the *API-SCmethods* component (as shown in Fig. 1) and relayed to the *DApp*.

In the following sections, we describe how a BA, working within the context of an SME, creates a simple smart contract to track activities, document flows, and express the business logic decisions of BPMN task elements using DMN, supported by FEEL.

IV. Documenting Flow of Information

The previous discussion of the BPMN model in Fig. 2 illustrates the flow of computation, which is forked by a fork-gate, enabling the concurrent execution of the *GetIns* and *GetTransp* tasks. The figure also shows how information flows along the computational process. This is achieved through the Business Analyst (BA) documenting the transfer of information between tasks using an association object. In Fig. 2, the dotted arrows, from the *RecAgr* task to the *SalesAgr* association object and then from the *SalesAgr* to the *GetTrReq* task, indicate the transfer of the sales agreement information (*SalesAgr*) from the *RecAgr* task to the *GetTrReq* task.

We first describe how JSON is used to model the flow of information and then provide simple examples to clarify. To provide more details on the content of the *SalesAgr* document, the BA clicks on *the SalesAgr* icon to provide annotation about its contents.

Information flowing along the computation process may contain multiple items, each of which is described by an array of key-value pairs. For this purpose, the BA uses JSON to represent the information flowing along the computation process. Items, such as *item1* and *item2,* are represented as an array of JSON elements.

The first element in the array has the form: { "source": "string1" }. The value of "string1" can only be "file" or "http", denoting whether the information is sourced from a file or an HTTP service. If the value of string1, representing the value for the key “source”, is "file", the next item in the array specifies the *CID* (Content Identifier) of the file from which the information is retrieved. This file is assumed to be in JSON format. The subsequent items in the array identify the fields (or components) within the file that need to be retrieved and passed as parameters to a smart contract method invoked by the *API-SCmethods* component.

If the value of string1 is "http", then there is an array of elements that contain information on (i) HTTP address of the service, (ii) input parameters, and (iii) output parameters containing the results of the service execution. The HTTP service is invoked with input parameters described, wherein the service returns information in its output parameters. Both the input and output parameters are described using the array elements. The HTTP service is invoked to implement the task and return the produced results in its output parameters that are recorded in the smart contract. For brevity, we will focus on describing how JSON is used to represent the content of files that provide information flowing along the computation process.

In Fig. 3, the file containing the relevant information is named *SalesAgr.json*, and its CID is provided. The array elements within the JSON structure identify which components of the *SalesAgr.json* file are to be retrieved and passed as parameters to the smart contract method. In our simple use case, the JSON components to be retrieved and passed to the smart contract method include only the *product ID*, which is supplied to both *GetTrReq* and *GetIns* tasks. These tasks then use this information to retrieve further details about the product to be transported.

This approach allows the BA to clearly define the flow of data and decision logic in the smart contract system, ensuring smooth interaction between the BPMN and DMN models, and providing transparency and traceability in the overall process.

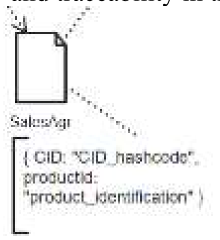

Fig. 3. Annotation to describe information flowing between tasks

Since a smart contract does not have direct access to external resources such as file systems or communication subsystems, information flowing out of a task is managed by the task emitting events that contain the information it produces. These emitted events are captured by the *API-SCmethods* component of the system architecture, which then stores them in a storage system. For simplicity, we assume this storage system is a file system.

On the other hand, information flowing into a task is prepared by the *API-SCmethods* component. It retrieves the information described by the JSON annotation of the *SalesAgr* association object, marshals it into the appropriate format, and passes it as input parameters to the smart contract method that implements the *GetTrReq* task.

For the subsequent sections, we assume that the monitor smart contract, required by the TABS+R tool, has already been deployed on the target blockchain. This can be either HLF blockchain or a blockchain that uses EVM.

V. Generating Smart Contracts by BA in SME

We analyzed a variety of use cases from the literature that focuses on transformation of BPMN models into smart contracts, such as use cases for Order-supply, Supply chains, Parts Orderr, Sales and Shipment, and Ordering Medications. In each case, document transfer between actors is central, but the creation, review, or amendment of these documents occurs off-chain. In such cases, the exchanged data between actors consists primarily of QR codes that identify the document files being shared, wherein the QR code is used as the documents unique that is analogous to the CID generated by the IPFS. The smart contract interactions among the partners are limited to the exchange of these documents, rather than directly handling the creation or modification of them.

Thus, when task executions can be performed off-chain, the task script code does not need to be provided on-chain, as long as the generation of the smart contracts from the BPMN model ensures a certified exchange of documents between on-chain and off-chain computations, which are readily supported by our approach as only CIDs are passed to the smart contract methods.

However, we wish to demonstrate also implementation of the business logic that is expressed using DMN that exploits the use of the FEEL language that has been designed by OMG to be understandable by both technical and non-technical people. For such cases, the BA uses DMN modeling with business logic expressed using the FEEL language. To express the DMN model and the business logic, we exploit the Camunda BPMN modeling language that enables expressing the DMN model and FEEL language statements for task element. Fig. 4 shows a portion of the BPMN model of Fig. 2 pertaining to the *DoTransp* task. The rectangular icon with rounded corners representing the task now has a picture of a small table of rows and columns in the left top corner – it indicates that the business logic modeled using DMN is to be implanted by the task.

Fig. 4 shows the use of association objects and their annotations interface used by the BA to describe the business logic that fails the whole smart contract if the cost of insurance is higher than the 15 percent of the sales price. For that purpose, information flowing into the task must include the purchase price from the *SalesAgr* and the cost of insurance from the insurance object/file identified by the "insurance_identification".

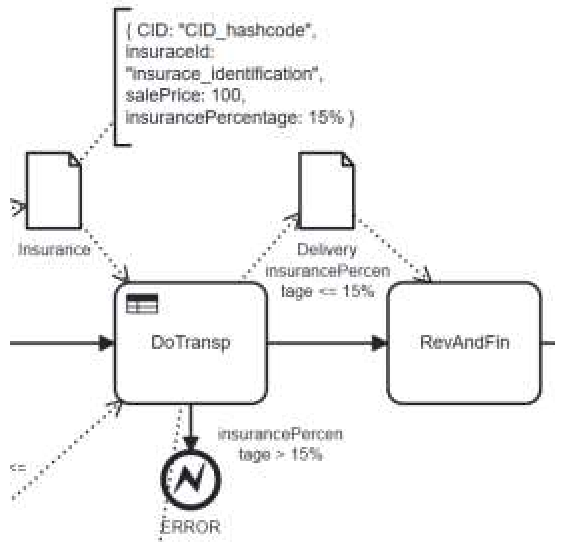
{ CID: "CID_hashcode",
insuraceId:
"insurace_identification",
salePrice: 100,
insurancePercentage: 15% }
Insurance
Delivery
insurancePercentage <= 15%
DoTransp
RevAndFin
insurancePercentage > 15%
ERROR

Fig. 4. DoTransp task and annotation of the information flowing along the flow of couputation

Operationally, in the absence of the IT support, the BA, or an operator trained by the BA, performs the actual activities represented by some of the tasks, while the smart contract records the result of the BAs activities. For instance, it is the BA who needs to execute the *GetTrReq* task. The BA needs the product description that is communicated by the BA to an insurance provider. The insurance provider communicates the insurance document to the BA who needs to store it in the file system to be accessible by the *API-SCmethods* component of the architecture shown in Fig. 1.

VI. Related Work

Several approaches to transforming BPMN models into smart contracts have been explored. The Lorikeet project [5] uses a two-phase methodology. It converts BPMN models into smart contract methods for the Ethereum blockchain plus create an off-chain component, which corresponds to our API-SCmethods component in Fig. 1, to support communication with a DApp. It also supports asset control, including tokens.

In contrast, the Caterpillar project [6, 7] focuses on recording all business processes within a single BPMN pool. Its architecture consists of three layers: Web Portal, Off-chain Runtime, and On-chain Runtime. It uses uses Ethereum for workflow control and management. Loukil et al. (2021) introduced CoBuP, which deploys a generic smart contract instead of directly compiling BPMN models [8]. CoBuP employs a three-layer architecture to convert BPMN into a JSON Workflow model for process execution on the blockchain. Similarly, Bagozi et al. (29) use a three-layer approach but in a simplified form, where BPMN models are first annotated to identify trust-demanding objects, followed by the creation and deployment of Abstract and Concrete Smart Contracts on a specific blockchain [15].

VII. Conclusions

In this paper we described how our tool, TABS+R, is used by a BA to facilitate generation of smart contracts from BPMN and DMN models without developer's assistance. Of course, this is only when the business logic can be expressed using DMN and FEEL. We are currently augmenting the tool to invoke HTTP services automatically to execute tasks, which would an approach for generation of smart contracts when smart contracts are generated in an organization that with IT services support HTTP services.

In addition, we also need to focuse on verifying and validating the security of the smart contract methods generated by our approach. Although we use standard techniques to secure individual smart contract methods, the concept of a long-running transaction enforced by an automatically generated transaction mechanism [11] requires protection from the man-in-the-middle attacks.

References

[1] D. Yang, C. Long, H. Xu, S. Peng, 2020. A Review on Scalability of Blockchain. 2nd ACM Int. Conf. on Blockchain Technology (ICBCT'20), pp 1–6. DOI:https://doi.org/10.1145/3390566.3391665

[2] P. J. Taylor, T. Dargahi, Dehghantanha, R. M. Parizi, 2019. A Systematic Lit. Review Of Blockchain Cyber Security – Science Direct. https://www.sciencedirect.com/science/article/pii/S2352864818301536.

[3] S. Khan, F. Loukil, C. Ghedira-Guegan, E. Benkhelifa, A. Bani-Hani, 2021. Blockchain smart contracts: Applications, challenges, and future trends. Peer Peer Netw Appl. 2021 Apr 18:1-25. doi: 10.1007/s12083-021-01127-0.

[4] O. Levasseur, M. Iqbal, and R. Matulevičius, 2021. "Survey of Model-Driven Engineering Techniques for Blockchain-Based Applications". PoEM'21 Forum: 14th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modelling.

[5] Tran, Q. Lu, and I. Weber, "Lorikeet: A Model-Driven Engineering Tool for Blockchain-Based Business Process Execution and Asset Management," in Proc. 2018 Int. Conf. on Business Process Management, 1–5; https://api.semanticscholar.org/CorpusID:52195200

[6] O. López-Pintado, L. García-Bañuelos, M. Dumas, I. Weber, and A. Ponomarev, "CATERPILLAR: A Business Process Execution Engine on the Ethereum Blockchain," Apr. 22, 2019, arXiv: arXiv:1808.03517. doi: 10.48550/arXiv.1808.03517.

[7] O. López-Pintado, M. Dumas, L. García-Bañuelos, and I. Weber, "Controlled flexibility in blockchain-based collaborative business processes," Information Systems, vol. 104, p. 101622, Feb. 2022, doi: 10.1016/j.is.2020.101622.

[8] F. Loukil, K. Boukadi, M. Abed, and C. Ghedira-Guegan. Decentralized collaborative business process execution using blockchain. World Wide Web, vol. 24, no. 5, pp. 1645–1663, Sep. 2021, doi: 10.1007/s11280-021-00901-7.

[9] BPMN, CMMN, and DMN Specifications at OMG. https://www.omg.org/intro/TripleCrown.pdf

[10] P. Bodorik, C. G. Liu, D. Jutla. 2022. TABS: Transforming Automatically BPMN Models into Smart Contracts. Blockchain: Research and Applications (Elsevier journal), 100115. https://doi.org/10.1016/j.bcra.2022.100115.

[11] C. Liu, P. Bodorik, D. Jutla. 2024. Tabs+: Transforming Automatically BPMN Models To Smart Contracts with Nested Collaborative Transactions. ACM journal on Distributed Ledger Technologies: Research and Practice (DLT) https://doi.org/10.1145/3654802.

[12] C. Liu, P. Bodorik, and D. Jutla, "Automated Mechanism to Support Trade Transactions in Smart Contracts". Revision submitted to Journal of Blockchain: Research and Applications, 2024. https://blockchain.cs.dal.ca/papers/BCRAj2wRepair-wPoC.pdf

[13] J. Benet. "IPFS - Content Addressed, Versioned, P2P File System." In https://github.com/ipfs/papers/raw/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf

[14] F. Loukil, K. Boukadi, M. Abed, and C. Ghedira-Guegan, "Decentralized collaborative business process execution using blockchain," World Wide Web, vol. 24, no. 5, pp. 1645–1663, Sep. 2021, doi: 10.1007/s11280-021-00901-7.

[15] Bagozi, D. Bianchini, V. De Antonellis, M. Garda, and M. Melchiori, "A Three-Layered Approach for Designing Smart Contracts. OTM 2019 Conf., Springer Int Publ, 440–457. doi: 10.1007/978-3-030-33246-4_28.