DEMO Models Based Automatic Smart Contract Generation: a case in Logistics using Hyperledger

David Aveiro
ARDITI - Regional Agency for the Development of Research Technology and Innovation & NOVA-LINCS Universidade NOVA de Lisboa & University of Madeira
Funchal, Portugal *daveiro@uma.pt*

Leonardo Abreu
ARDITI - Regional Agency for the Development of Research Technology and Innovation & University of Madeira
Funchal, Portugal *leonardo.tadeu@arditi.pt*

Duarte Pinto
ARDITI - Regional Agency for the Development of Research Technology and Innovation & University of Madeira
Funchal, Portugal *duarte.nuno@arditi.pt*

Vítor Freitas
ARDITI - Regional Agency for the Development of Research Technology and Innovation & University of Madeira
Funchal, Portugal *vitor.freitas@arditi.pt*

Abstract

This article presents a practical research project aimed at developing a method for automatically generating smart contracts from business models. The project has as a context the logistics industry and uses Hyperledger Fabric as the blockchain (BC) platform. The main contributions are a mapping from DEMO (Design and Engineering Methodology for Organizations) language to Hyperledger Chaincode using GO language, as well as an evolution of DEMO's Action Model Grammar, that enable specification of elements necessary for automatic SC generation. The proposed approach extends the DEMO methodology so that it includes an SC concern, enabling the generation of reusable action rule specifications and other elements necessary for SC generation. Our research contributes to combining the strengths of the DEMO methodology and smart contracts. The design and implementation considerations of this approach are discussed in detail, and the results can be applied in future business cases requiring enterprise interoperability supported by distributed ledger technology.

Keywords: Enterprise engineering, DEMO Action Model, Blockchain, Hyperledger Fabric, Smart Contracts, Logistics industry

1. Introduction

The emergence of Distributed Ledger Technology (DLT) has paved the way for revolutionary solutions like blockchain, which not only provide a distributed and decentralized database but also ensure synchronization and consensus among all parties involved[1]. The beauty of BC technology lies in its ability to provide an immutable ledger for decentralized transactions. Not only is it ideal for financial services like digital assets, remittance and online payment, but it has also found applications in diverse fields such as smart contracts, public services, Internet of Things (IoT), reputation systems, and security services. By leveraging the decentralized nature of blockchain, businesses can enjoy unparalleled reliability and honesty, while the tamperproof security of transactions ensures airtight protection against fraudulent activities. Furthermore, the automation of smart contracts is seamlessly enabled by the distributed nature of BC technology[2].

A smart contract (SC) is an autonomous and self-executing program that operates on top of a BC network. It is designed to enforce the terms and conditions of an agreement between parties without the need for intermediaries or centralized authorities. The terms of the contract are pre-coded and transparently recorded on the blockchain, ensuring that they are automatically and immutably executed. This is particularly beneficial for parties who may lack an established level of trust in their relationship, as the SC eliminates the need for intermediaries to enforce the agreement, providing a secure and efficient means of executing complex business processes [3].

Smart contracts based on BC technology have the potential to revolutionize logistics management by providing a secure, efficient, and transparent system for tracking goods and transactions, ultimately leading to improved supply chain management [4].

Hyperledger Fabric is an open-source BC platform that overcomes the limitations of traditional permissioned blockchains. It is used in more than 400 prototypes, proofs-of-concept, and production distributed ledger systems, across different industries and use cases. Fabric introduces a new BC architecture aiming at resiliency, flexibility, scalability, and confidentiality [5].

In Enterprise Engineering (EE), recent progress involves integrating DEMO (Design and Engineering Methodology for Organizations) models with proof-of-concept SC implementations. This emphasizes synergizing Construction and Action Models of DEMO with BC technology for secure notarization and transaction execution. [6, 7].

By leveraging the benefits of BC technology, such as immutability and transparency, while simultaneously utilizing the modelling and analysis capabilities of the DEMO methodology, this approach can lead to more efficient and effective execution of business processes. In particular, this development has significant implications for the design and implementation of smart contract-enabled systems in enterprise-level applications, as it enhances interoperability and scalability [8].

Despite recent advancements in the integration of the DEMO methodology with SC implementations, current research suggests that the generation of smart contracts from DEMO models (as well as models from other modelling languages) still heavily relies on manual coding. This situation limits the potential of Distributed Ledger Technologies (DLTs) in the implementation of DEMO-based information systems, as it makes the process of contract production a primarily manual endeavor [9, 10].

Although there have been notable advancements in the integration of the DEMO methodology with SC implementations, there is still a significant need to evolve the DEMO specification language. The goal is to enable the automatic generation of DEMO-based smart contracts with minimal manual coding, or even eliminate the need for manual coding altogether. This evolution would further enhance the potential of Distributed Ledger Technologies (DLTs) in the implementation of DEMO-based information systems by significantly reducing the human effort and potential for error in contract production.

Recent studies [11, 12] have illustrated that DEMO extensions have the potential to capture the essence of an organization, even if they are not semantically equivalent to DEMOv4.5 [13].

It appears that these extensions have the potential to offer a comprehensive perspective that not only enables integration with BC technology, but also facilitates communication by aligning with the language of business practitioners. Based on this premise, the research question addressed in this paper is as follows: Can we evolve DEMO's Action Model Grammar to enable fully automatic smart contract generation?

2. Literature Review

In today's fast-paced global economy, companies in the logistics and manufacturing industries face the challenge of meeting increasing demand for greater transparency in their supply chains. Customers and regulators alike demand a full understanding of where products come from, how they are made, and how they are transported from one point to another. This requires comprehensive tracking solutions that enable traceability from start to finish. Centralized tracking solutions have been the traditional approach to supply chain transparency, but they may not be the most efficient or effective. Decentralized concepts for tracking goods and transport containers offer a more reliable, secure, and transparent approach. Decentralization increases data security, reduces data errors, and eliminates single points of failure in the tracking process.Implementing decentralized tracking solutions can be challenging for IT teams, requiring comprehensive technical solutions to ensure secure data access across multiple locations. Despite this, the benefits of decentralized tracking solutions outweigh the drawbacks. Companies that implement them can provide real-time information to customers, reduce counterfeiting and fraud risks, and ensure regulatory compliance[14].

Public blockchains involve a native cryptocurrency and often use consensus based on "proof of work" (PoW) and economic incentives. Permissioned blockchains run a BC among a set of known, identified participants, providing a way to secure the interactions among a group of entities that have a common goal but do not fully trust each other.

Fabric is the first BC system to support the execution of distributed applications written in standard programming languages, in a way that allows them to be executed consistently across many nodes. The architecture of Fabric follows a novel execute-order-validate paradigm for distributed execution of untrusted code in an untrusted environment. It separates the transaction flow into three steps: executing a transaction and checking its correctness, ordering through a consensus protocol, and transaction validation per application-specific trust assumptions [5].

Creating smart contracts can be a complicated and time-consuming process, requiring a deep understanding of programming languages and BC frameworks. In response, Choudhury et al. (2018) proposed a method that leverages domain-specific ontologies and semantic rules to automate the translation of constraints from legal agreements or protocols to smart contracts.

This innovative method uses parsing and abstract syntax tree manipulation techniques to generate smart contracts automatically based on specific requirements, facilitating the translation of constraints from legal agreements or protocols and enabling reproducibility. The effectiveness of the approach was demonstrated through two use cases: clinical trial protocols and car rental criteria.

The proposed method has the potential to be extended to other programming languages and BC frameworks, making it an attractive option for businesses and individuals seeking to streamline the SC creation process. By automating SC creation, the process becomes more efficient and accessible, creating new opportunities for businesses and individuals to benefit from smart contracts [9].

Aparicio et al. [6] proposes a novel method to use ontological models for supporting the accurate implementation of Smart Contracts in the BC of an organization. The authors showcase the feasibility of this method using the Rent-A-Car case study. The paper also explores the correlation between BC Smart Contracts and DEMO Action Model and how to generate the latter from the former. The main contribution of the paper is the automatic extraction of knowledge from the DEMO Action Model for producing Smart Contracts in Blockchain. This approach leverages the ontological modelling done for the correct implementation of Smart Contracts, resulting in increased efficiency and consistency. The authors demonstrate the effectiveness of this approach by using a real-world case study.

In a newer publication, Aparicio et al. [7] reinforce the idea of use of BC technology is explored as a means of coordinating inter-organizational processes involving untrusted parties.

Specifically, the paper proposes the use of smart contracts to implement DEMO Action Models and discusses how model-driven engineering can be used to automatically generate these contracts. The paper also highlights the potential benefits of using ontology-based modelling methods to aid in the formal specification, inference, and verification of smart contracts. The authors note that the proposed mapping between Solidity Concepts and DEMO Action Model concepts can be seen as a way to implement DEMO Action Models, but optimization was not addressed in this work and is left for future research. The Rent-A-Car Use Case is used to verify the research, but the authors suggest that further validation is needed through testing the mapping proposed in a larger sample of Action Models. Overall, the paper's findings suggest that modelling methods based on ontology can lead to better data standards, business practices, and processes for developing and operating a blockchain. This is particularly relevant in the context of smart contracts, which are pieces of software that represent a business arrangement and execute themselves automatically under predetermined circumstances. The authors argue that ontology-based BC modelling can enhance interpretability, which could lead to a new level of security for BC systems.

Tong et al. (2022) [15] have proposed a novel idea for the automatic generation of smart contracts. They propose a framework called AI-assisted SC Generation (AIASCG), which allows contracting parties in heterogeneous contexts and different languages to collaborate and negotiate the contract clauses. This proposal provides a universal representation of contracts through machine natural language (MNL) as the common understanding of the contract obligations. The authors propose an AI-based automatic word segmentation technique called Separation Inference (SpIn) to automatically split the sentence. SpIn serves as the core component in AIASCG, accurately recommending the intermediate MNL outputs from a natural language sentence and tremendously reducing the manual effort in contract generation.

3. Proposed approach

3.1. Description of the case study in logistics

Micro-hubs are strategic logistics centers that promote collaboration among express and freight forwarders, improving service quality and geographic coverage while minimizing infrastructure investments. Trust is a key challenge in data exchange, but with the right technology, a more efficient and sustainable logistics industry can be built.

This paper discusses the LOGHL (LOGistics HyperLedger project – fictitious name for blind review purposes) project, an innovative micro-hubs initiative aimed at addressing trust and collaboration challenges in the logistics industry using blockchain (BC) technology. LOGHL aims to overcome barriers to entry for new participants in collaborative micro-hubs by implementing a reliable digital solution. The proposed solution utilizes smart contracts to record all micro-hub transactions in a verifiable, permanent, and transparent manner. By doing so, all stakeholders, including express couriers, producers, end consumers, and recycling centres, can access accurate and reliable information about the transactions, fostering trust and enabling collaboration towards common objectives. Additionally, LOGHL incorporates a circular economy component to promote sustainable practices.

Overall, the LOGHL project represents an exciting and innovative approach to addressing some of the biggest challenges facing the logistics industry today. By leveraging cutting-edge technology like BC and smart contracts, LOGHL has the potential to transform the way logistics operators work together and create a more sustainable and efficient industry for everyone involved.

In a recent study by Tallyn et al. [16], they examined the use of smart-contracts to coordinate four different urban delivery scenarios (Person-to-Person, Hub-to-Person, Person-to-Hub, and Hub-to-Hub) with logistics professionals utilizing their existing work practices. The study highlights the balance between automation and the need for social connections to maintain trust between couriers and recipients in the last-mile delivery process, especially when hubs and lock-boxes are introduced. The study demonstrates the potential advantages of smart contracts in enhancing the delivery process, while recognizing the significance of social connections for building trust between couriers and recipients.

Kim and Laskowsky et al. [17] were inspired by the blockchain-design used in the TOVE traceability ontology and created smart contracts aimed at tracking physical resources and their origin across international supply chains. The execution of these contracts takes into account the ontological traceability constraints that are enforced on the distributed ledger. Kim and Laskowsky's work demonstrates how smart contracts can be designed to facilitate the tracing of physical resources and their provenance across complex supply chains, while taking into account ontological traceability constraints.

While there has been some exciting work exploring the potential applications of BC in collaborative micro-hubs, the current literature falls short in providing practical solutions that respond to the unique logistics needs of circular economy processes. To truly realize the full potential of this innovative approach, we need a viable and scalable platform that can generate and execute smart contracts between all the different players involved in a collaborative micro-hub network.

3.2. Overview of the DEMO's Action Model used for SC generation

Information system implementation aims to improve organizational performance, but success depends on factors such as system capabilities and organizational characteristics. DEMO utilizes the PSI theory of Enterprise Ontology to examine an organization's essence and creates interconnected models and diagrams for an accurate representation of reality [18]. We propose a new approach to SC development from DEMO models, making it less error-prone and more efficient. With a recent proposal of DEMO's action rule specification (ARS) language, it is possible to provide a more robust and formal specification of logical and mathematical expressions, as well as flow control of business rules [18]. This level of precision is crucial for smart contracts, where even the slightest error could result in costly consequences.

Moreover, our approach caters to a wider audience by allowing the specification of rules in a visual programming language. This not only makes it easier for non-technically savvy collaborators to participate in the process, but also enhances the overall effectiveness of the development cycle. By leveraging this powerful combination of formal language and visual programming, We're confident that our proof of concept is a solid start to develop blockchain smart contracts and has the potential to significantly boost any smart contract project in the future.

The new ARS language offers a unique advantage by supporting constructs and rules with references to features such as smart contracts, documents, and templates. This, in turn, has a significant impact on the essential models built on top of the AM, bringing their implementation closer to a real-world enterprise information system (EIS). The integration of these key features allows for a more robust and versatile system that can cater to the needs of a diverse range of businesses. The ARS language appears to be a promising option for those seeking to develop efficient and dependable enterprise information systems.

3.3. From DEMO Action Rule Models to SC in GO

In this section, we will discuss the process of transitioning from DEMO Action Rule Models to Hyperledger Chaincode Go. As the details of the implementation involve a considerable amount of code, we have created a public Google Drive link [19] where you can find all the relevant code. This allows the reader to analyse the implementation in detail.

Essential Methods for our Framework

To streamline the transpiling of DEMO language to Chaincode implemented in GO language, several default methods have been developed. These methods will be integral to all smart contracts automatically generated by our framework presented in this paper.

CreateCompositeKey

The first method creates a composite key for a specific entity in the context of a transaction in a smart contract. The composite key is created by concatenating a string that represents the entity and a list of strings that represent the entity's identifiers. The composite key is a way to efficiently index, and search records stored in the ledger of a blockchain. When the composite key is created, the BC indexing system stores the key in an index table. This allows queries for specific entities and associated identifiers to be executed more quickly, as the data is already indexed and can be accessed quickly by the system. The utility of this method is to provide a convenient and efficient way to create composite keys in smart contracts and facilitate the indexing of records on the blockchain. This optimizes access to stored data, allowing for better performance and scalability of BC applications.

UpsertEntityRecord

The purpose of the "UpsertEntityRecord" method is to insert or update an entity record in a distributed ledger, which is managed by the BC platform on which the contract is being executed. This method is useful for decentralized applications that need to store data in a distributed ledger and ensure that the data is immutable and secure. By using a SC to manage access to the ledger, it is possible to ensure the integrity and validity of the data, as well as the security of the transactions.

EntityRecordExists

The method "EntityRecordExists" is a function defined in an SC that checks if a record with the specified key exists in the world state of the blockchain. This function can be useful in various BC applications, as it allows verifying if a record with a specific key already exists in the world state of the BC before attempting to create a new record with the same key. This helps to avoid conflicts and ensure the integrity of data stored on the blockchain.

CreateEntityIterator

The "CreateEntityIterator" method is a function that is part of a SC and is designed to create an iterator for retrieving data of a specific entity type stored on the blockchain. In the context of an auction, for example, this method can be used to retrieve all bids made for a particular auction (identified by its ID) on the blockchain. The auction ID would be passed as a parameter to the function, which would return an iterator with all the corresponding bids.

StartTransaction & CloseTransaction

The "StartTransaction" function is used to initiate a transaction and generate a unique ID for it using the GetTxID method of the stub (a data handling object provided by the ContractAPI). The purpose of this function is to provide a starting point for the execution of the smart contract's business logic. It also helps track the transaction and identify the exact moment it started. The "CloseTransaction" function is used to finalize the transaction. If an error occurs during the execution of the transaction, the function receives a boolean value err equal to true and performs a rollback of the transaction, If there are no errors, the function emits a "tx.commit" event to indicate that the transaction was successful. The purpose of this function is to ensure that the transaction is executed safely and consistently, and that the state of the SC is updated correctly. It also helps notify other stakeholders about the result of the transaction and maintain a record of successful and failed transactions.

Automated SC Generation using DEMO's Action Model: Our Approach

To address the research question that aims to validate if a specification of the DEMO language, based on an expansion of it, is capable of generating Hyperledger Smart Contracts, the focus was on three simple transactions in the LOGHL project: T01 – "Parcel Delivery Add Parcel", T02 – "Parcel Delivery Auction Start" and T03 – "Parcel Delivery Bid"

The T01, T02, and T03 transactions serve specific functions in a delivery platform. The GO code and DEMO language specifications for each transaction were generated, and the next step was to use reverse engineering to map the languages from one to the other.

Reverse engineering involves analyzing a system to determine its components and their relationships to extract useful information. By analyzing the GO code and DEMO language specifications, it may be possible to identify common elements and patterns that can be used to map from one language to the other. This can provide valuable insights into how the two languages relate to each other and can help refine the process of generating smart contracts from DEMO specifications.

Overall, reverse engineering can be a useful tool for understanding and mapping between different languages and systems, especially when relationships between components are not immediately apparent.

DEMO to Hyperledger GO Chaincode Transpiling Method

To create a smart contract that generates a ledger entry, the first step is to extract relevant information from the DEMO specification. This involves identifying all the entities that will be part of the smart contract based on the Entity specification in DEMO language, and converting them into Go structs.

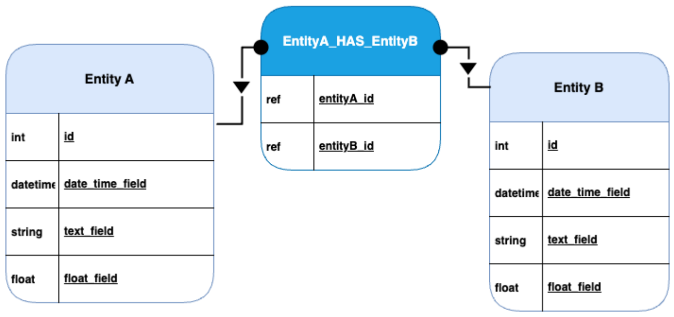


Fig. 1. General Fact Modal

To store assets in a struct on the ledger, it is necessary to translate the DEMO Fact Model to Chaincode in Go. This can be achieved through straightforward mapping, with some adjustments as necessary.

We can use the Fact Model shown in Fig. 1 to demonstrate how this mapping is done. By examining the model, we can quickly understand how it can be translated into Go code for use in our smart contract,

Listing 1. Creating Go structs based on Fact Model

```
type EntityA struct {
  ID             int      `json:"id"`
  DATE_TIME_FIELD time.Time `json:"date_time_field"`
  TEXT_FIELD   string  `json:"text_field"`
  FLOAT_FIELD float32  `json:"float_field"`
}
type EntityA_HAS_ENTITYB struct {
  ENTITY_A_ID int `json:"entity_A_id"`
  ENTITY_B_ID int `json:"entity_B_id"`
}
```

Once the entities are identified, the next step is to map all the actions specified in the DEMO specification to the corresponding Chaincode. This mapping enables implementation of a process for transpiling between DEMO and Chaincode, enabling the generation of correct smart contracts.

The SC method name is derived from the pattern used on Action Rules in the Grammar. We will start with a specific example and then generalize it to present our proposal of a general method.

In Fig. 2, the first block shows the "WHEN transaction_type IS | HAS_BEEN transaction_state action" DEMO specification for an Action Rule, which is flagged for SC generation. When the Chaincode is generated, this rule is translated to:

Listing 2. DEMO Action Rule "WHEN" to Chaincode Go

```
func (s *SmartContract) transaction_type(ctx
    contractapi.TransactionContextInterface)(bool, error) {
   // Smart Contract Code
}
```

The primary goal of this part of our proposal is to facilitate interaction with the BC ledger. To achieve this goal, the "user input single entity type" action is implemented. The "User input single entity type" is a block that specifies that, in the context of a system's execution, the user is requested, through a form, to manually fill in a set of data. As it involves human input, several validations are necessary to avoid inconsistencies or incorrect data in the BC. The specification for this Action Rule is ACTION "user input single entity type" SCOPE Entity entityName PROPERTY(IES) PROPERTY. The corresponding Chaincode for this will generate the following code:

Listing 3. Converting user input single entity type to Chaincode in Go using DEMO Action Rule

```
func (s *SmartContract) transaction_type(ctx
    contractapi.TransactionContextInterface, entity
    models.entityName)(bool, error) {
     entityKey, err := s.CreateCompositeKey(ctx, EntityName,
         []string{fmt.Sprint(entity.ID)})
     if err != nil {
        return false, err
       }
  if err := validateentityName(&entity); err != nil {
     return false, err
  }
  _, err = s.UpsertEntityRecord(ctx, entityKey, entity)
  if err != nil {
     return false, err
  }
       return true, nil
}
```

The function "validateentityName" shown on the previous code, is created based on property validation condition on DEMO Action Rule specification, as you can see, on the example, but first we need to specify how "IF condition THEN action [ELSE action]" grammar is translated to Chaincode.

The "validateentityName" function shown in the previous code is created based on the property validation condition specified in the DEMO Action Rule specification. As you can see from the example, this function is called based on the condition specified in the IF-THEN-ELSE grammar.

Before discussing the function further, it's important to specify how the "IF condition THEN action [ELSE action]" grammar can be translated to Chaincode.

Listing 4. Converting "IF condition THEN action ELSE action" DEMO Action Rule Grammar to Chaincode Go

```
   if(condition){
      // Perform Action inside THEN
   } else{
      // Perform Action inside ELSE
   }
```

With this understanding, we can now better interpret the validation condition of a property. The condition is interpreted as an "IF condition THEN action [ELSE action]" grammar. However, it can be simplified further. If the condition evaluates to true, then the validation criteria doesn't match, and an error message is generated based on the property name and condition.

For example, if we have a property of type "int", it must be greater than '0'. Therefore, if the condition is not met, an error message will be generated.

Listing 5. Creating validation method to user input

```
func validatetransaction_type(entity *models.EntityName) error {
   var errorMessages []string
   if !(entity.propertyName > 0) {
      errorMessages =
         append(errorMessages, fmt.Sprintf("%s most be bigger than
            %d ", entity.propertyName, 0))
   }
   // Validate all properties of the entity
   (...)
   if len(errorMessages) > 0 {
    return errors.New(strings.Join(errorMessages, "\n"))
   }
   return nil
}
```

We are considering an extension to our grammar that allows custom error messages to be generate.This will enable us to provide more informative error messages to users, at this stage error messages are generated based on property name and condition defined on DEMO specification.

Having a solid foundation for creating and validating assets on the ledger we can now dive in more complex transaction types: sometimes a method needs to add more than one asset on the ledger, in these cases, we need to be sure that either all transaction actions occur or the changes made are rolled back for the previous state. In order to achieve this kind of behaviour, when a single action rule has more than one Action of type "user input single entity type" then a transaction-based approach is used, the Chaincode can maintain the integrity of the ledger by either committing all changes made within the transaction or rolling them back if an error occurs. This ensures that the data stored on the ledger is accurate and complete, and can be trusted by all parties involved in the system.

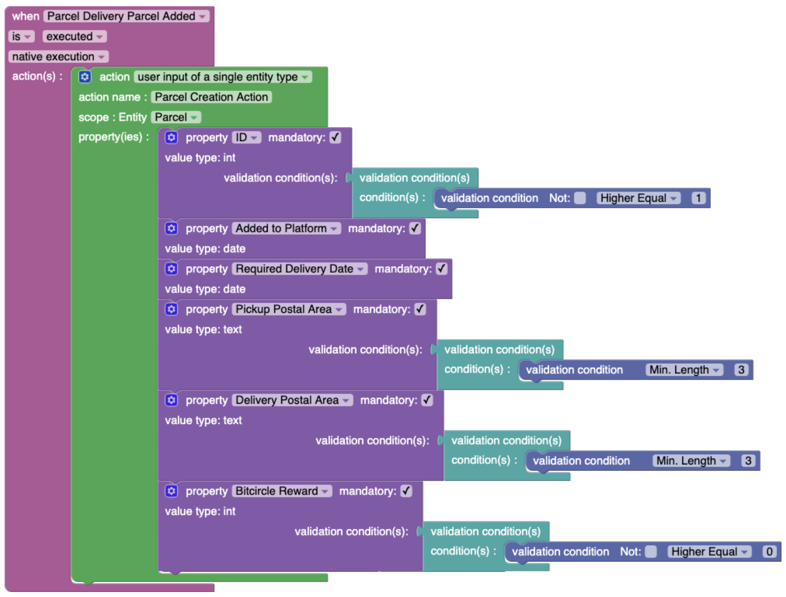
In cases where complex logic is required, such as finding a specific record on the ledger by iterating over a group of assets, the DEMO Action Rule specification uses the "FOREACH EntityType entity set action" construct. This represents a modification that needs to be made to our grammar.

Previously, when using DEMO to generate smart contracts, there was no need to identify the Entity Type that is being iterated. However, with the introduction of the "FOREACH" construct, this becomes necessary. This modification allows us to handle more complex logic within our smart contract generation process. General creation of this kind of loop is represented next.

Listing 6. DEMO "FOREACH" implementation using Chaincode

```
for _, entity := range entitySlice {
   // CODE HERE
}
```

A practical application of these rules can be seen on a public Google Drive link [19]. Here, you can find examples that illustrate the conversion process between DEMO specifications and Hyperledger GO Chaincode. This provides a concrete demonstration of how the rules can be used to generate functional smart contracts from DEMO specifications.



```
when Parcel Delivery Parcel Added
is executed
native execution
action(s) : action user input of a single entity type
            action name : Parcel Creation Action
            scope : Entity Parcel
            property(ies) : property ID mandatory: ✓
                            value type: int
                                validation condition(s): validation condition(s)
                                    condition(s) : validation condition Not: ☐ Higher Equal 1
                            property Added to Platform mandatory: ✓
                            value type: date
                            property Required Delivery Date mandatory: ✓
                            value type: date
                            property Pickup Postal Area mandatory: ✓
                            value type: text
                                validation condition(s): validation condition(s)
                                    condition(s) : validation condition Min. Length 3
                            property Delivery Postal Area mandatory: ✓
                            value type: text
                                validation condition(s): validation condition(s)
                                    condition(s) : validation condition Min. Length 3
                            property Bitcircle Reward mandatory: ✓
                            value type: int
                                validation condition(s): validation condition(s)
                                    condition(s) : validation condition Not: ☐ Higher Equal 0
```

Fig. 2. Action Rule Demo example

4. Results and Discussion

The results of this research demonstrate that it is possible to automatically generate Smart Contracts for Hyperledger Fabric using DEMO Action Rule Models, which has several advantages, including the reduction of time and resources required to create complex contracts, and the ability to automate data validation, error management, and interaction between data on the blockchain.

Three basic transactions (create parcel, create auction, place bid) were developed in Smart Contracts using Go, and through reverse engineering, it was possible to confirm that automatic SC generation is feasible. Several cases of automatic generation were validated, including data validation, relationship between data, iterations between new transactions and data present in the blockchain, BC searches, and error management. The results suggest that this approach can be an efficient way to develop Smart Contracts, reducing the time and resources needed to create complex contracts.

However, the use of DEMO may also present some limitations in the expressiveness of the DEMO language, which may restrict the ability to create smart contracts with more advanced logic. This leads to the creation of new components of the DEMO language to solve specific problems related to the blockchain. By making these changes, the proof of concept can evolve into a more comprehensive solution, allowing for the development of smart contracts with greater complexity and versatility in the blockchain environment.

5. Conclusions and Future Work

The results of this research are highly promising and indicate that the use of DEMO Action Rule Models for the automatic generation of Smart Contracts can significantly reduce the time and resources required to develop complex contracts. This approach will also facilitate the creation of more efficient and secure contracts by automating data validation, error management, and interaction between data on the blockchain.

Furthermore, the potential applications of this approach are vast, as it can be extended to other business areas as long as there is an appropriate DEMO model to represent the business process. This can lead to significant advancements in the development of smart contracts and the wider adoption of BC technology.

For future research directions, an interesting area to explore would be the inclusion of more advanced functionalities in automatically generated contracts, thus simulating an entire logistics business covering all its needs. Regarding the possibility of extending DEMO contracts to other business areas, this approach may be suitable for other domains as long as there is an appropriate DEMO model to represent the business process. However, the use of DEMO to address specific topics may need to be supplemented with new grammatical rules, and the best approach for creating smart contracts should be evaluated on a case-by-case basis. Overall, these results represent a significant contribution to the field of SC development, especially in the context of BC and Hyperledger Fabric.

In terms of future work, a low-code platform based on DEMO currently being developed [20][21] will implement a component that transpiles DEMO models into GO code, generating the necessary code for blockchain implementation. This will also allow for the definition of the participating entities architecture, as well as the automatic generation of code that implements the necessary APIs.

Overall, the use of DEMO Action Rule Models could represent a game-changing development in the field of SC development, particularly in the context of BC and Hyperledger Fabric. The benefits of this approach are clear, and future research should continue to explore ways to extend the functionalities of automatically generated contracts and evaluate the best approach for creating smart contracts on a case-by-case basis.

References

1. S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” *Cryptography Mailing list at https://metzdowd.com*, Mar. 2009.
2. Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, “An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends,” in *2017 IEEE International Congress on Big Data (BigData Congress)*, pp. 557–564, June 2017.
3. N. Szabo, “Smart Contracts : Building Blocks for Digital Markets,” 2018.
4. N. Álvarez Díaz, J. Herrera-Joancomartí, and P. Caballero-Gil, “Smart contracts based on blockchain for logistics management,” in *Proceedings of the 1st International Conference on Internet of Things and Machine Learning*, IML ’17, (New York, NY, USA), pp. 1–8, Association for Computing Machinery, Oct. 2017.
5. E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, “Hyperledger fabric: a distributed operating system for per-

missioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, (New York, NY, USA), pp. 1–15, Association for Computing Machinery, 2018.

6. M. Aparício, S. Guerreiro, and P. Sousa, "Automated DEMO Action Model Implementation using Blockchain Smart Contracts," Nov. 2020.
7. M. Aparício, S. Guerreiro, and P. Sousa, "Decentralized Enforcement of DEMO Action Rules Using Blockchain Smart Contracts," Mar. 2021.
8. B. Hornáčková, M. Skotnica, and R. Pergl, "Exploring a Role of Blockchain Smart Contracts in Enterprise Engineering: 8th Enterprise Engineering Working Conference, EEWC 2018, Luxembourg, Luxembourg, May 28 – June 1, 2018, Proceedings," pp. 113–127, Jan. 2019.
9. O. Choudhury, N. Rudolph, I. Sylla, N. Fairoza, and A. Das, "Auto-Generation of Smart Contracts from Domain-Specific Ontologies and Semantic Rules," Sept. 2018.
10. M. Skotnica and R. Pergl, "Das Contract - A Visual Domain Specific Language for Modeling Blockchain Smart Contracts," pp. 149–166, Jan. 2020.
11. D. Pacheco, D. Aveiro, D. Pinto, and B. Gouveia, "Towards the X-Theory: An Evaluation of the Perceived Quality and Functionality of DEMO's Process Model," in *Advances in Enterprise Engineering XV* (D. Aveiro, H. A. Proper, S. Guerreiro, and M. de Vries, eds.), Lecture Notes in Business Information Processing, (Cham), pp. 129–148, Springer International Publishing, 2022.
12. D. Pacheco, D. Aveiro, B. Gouveia, and D. Pinto, "Evaluation of the Perceived Quality and Functionality of Fact Model Diagrams in DEMO," in *Advances in Enterprise Engineering XV* (D. Aveiro, H. A. Proper, S. Guerreiro, and M. de Vries, eds.), Lecture Notes in Business Information Processing, (Cham), pp. 114–128, Springer International Publishing, 2022.
13. "2020-07-31 DEMO Specification Language 4.5 – Enterprise Engineering Institute."
14. E. Tijan, S. Aksentijevic, K. Ivanić, and M. Jardas, "Blockchain Technology Implementation in Logistics," *Sustainability*, vol. 11, p. 1185, Feb. 2019.
15. Y. Tong, W. Tan, J. Guo, B. Shen, P. Qin, and S. Zhuo, "Smart Contract Generation Assisted by AI-Based Word Segmentation," *Applied Sciences*, vol. 12, p. 4773, Jan. 2022. Number: 9 Publisher: Multidisciplinary Digital Publishing Institute.
16. E. Tallyn, J. Revans, E. Morgan, K. Fisken, and D. Murray-Rust, "Enacting the Last Mile: Experiences of Smart Contracts in Courier Deliveries," pp. 1–14, May 2021.
17. H. Kim and M. Laskowski, "Toward an ontology-driven blockchain design for supply-chain provenance," *Intelligent Systems in Accounting, Finance and Management*, vol. 25, pp. 18–27, Jan. 2018.
18. D. Aveiro and J. Oliveira, "Towards DEMO model-based automatic generation of smart contracts," in *Advances in Enterprise Engineering XVI - 12th Enterprise Engineering Working Conference, EEWC 2022, Leusden, The Netherlands, November 2-3, 2022* (e. a. Cristine Griffo, ed.), vol. 473 of *LNBIP*, pp. 71–89, Springer, 2022.
19. "DEMO Models Based Automatic Smart Contract Generation: a case in Logistics using Hyperledger," May 2023. url:https://bit.ly/LOGHLproj original-date: 2023-05-01.
20. V. Freitas, D. Pinto, V. Caires, L. Tadeu, and D. Aveiro, "The DISME low-code platform - from simple diagram creation to system execution, in CEUR-WS.org/Vol-3388 - EEWC Forum & EEWC Doctoral Consortium 2022 & EEWC Posters, colocated with the 12th Enterprise Engineering Working Conference (2022)," May 2023.
21. D. Aveiro and V. Freitas, "A new action meta-model and grammar for a DEMO based low-code platform rules processing engine," in *Advances in Enterprise Engineering XVI - 12th Enterprise Engineering Working Conference, EEWC 2022, Leusden, The Netherlands, November 2-3, 2022* (e. a. Cristine Griffo, ed.), vol. 473 of *LNBIP*, pp. 33–52, Springer, 2022.