A Smart Contract Development Framework for Maritime Transportation Systems

Xufeng Zhao[1,2], Qiuyang Wei[1,3], Xue-Yang Zhu[1,2] and Wenhui Zhang[1]
[1]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[2]University of the Chinese Academy of Sciences, Beijing, China
[3]Hangzhou Institute for Advanced Study, University of Chinese Academy of Sciences, Hangzhou, China
{zhaoxf, weiqy, zxy, zwh}@ios.ac.cn

Abstract—A smart contract is a computerized protocol running on the blockchain, which provides a reliable environment for transactions among trustless participants. The business of maritime transportation is usually with multifarious participants and burdensome paperwork. Blockchain-based smart contract systems are promising to improve the efficiency and transparency of the transactions. However, due to the gap between domain experts and programmers, and the diversity of maritime business scenarios, it is challenging to efficiently develop reliable smart contracts for this domain. In this paper, we propose MariSmart, a novel development framework for maritime transportation smart contracts, which consists of a business logic model and a set of Solidity templates. The templates are designed based on the business logic model extracted from the domain knowledge. We carry out case studies on three real-world maritime transportation systems to show the feasibility and effectiveness of the framework.

Keywords–Smart Contract; Maritime Transportation; Supply Chain; Blockchain.

1. Introduction

A smart contract was defined as 'a computerized transaction protocol that executes the terms of a contract' by Nick Szabo [1] in 1994. The concept of smart contract has gained much attention since the emergence of blockchain technology. Blockchain platforms, such as Ethereum [2], provide a reliable environment for smart contract execution. Due to the decentralized, tamper-proof, and indisputable nature of the blockchain, smart contracts can perform reliable transactions among trustless participants. Blockchain and smart contract technologies have gained interest from various fields, such as finance [3][4], healthcare [5][6][7], supply chain [8][9] and transportation [10][11][12].

Businesses of the maritime transportation are usually international. There are multiple stakeholders participating in transactions, who do not necessarily trust each other. Also, there is much time-consuming paperwork in this domain. For example, a flower export consignment from Kenya to the Netherlands contains more than 200 bilateral communications among 20 organizations, and it usually takes about 10 days to finish the paperwork [13]. It's well acknowledged that blockchain-based smart contract systems can improve the efficiency and transparency of transportation, build trust among stakeholders and expand cooperation [13] [14] [15].

To efficiently develop blockchain-based smart contract systems for maritime transportation, the first challenge is how to increase code reusage. Due to the lack of consensus between domain experts and programmers, developing smart contracts from scratch is rather demanding and inefficient. For example, maritime transportation smart contracts proposed in [11] [12] [16] are for a similar goal, yet their implementations vary from each other, and therefore can merely be reused. We address this challenge by proposing a set of smart contract templates for the maritime transportation domain, which can be inherited by particular smart contracts so that developers only have to implement incremental functionality.

Another challenge that rises with template-based development is that the framework has to be compatible with different business scenarios. Since the stakeholders, workflow of maritime transportation vary in different scenarios, and there is a lack of a universal business model in the maritime sector, it's challenging to design extendable templates for multiple possible uses. We address this challenge by proposing a business logic model to guide the design of templates.

In this paper, we propose **MariSmart**, a development framework for smart contracts that monitor and manage maritime transportation business. The framework is shown in Figure 1. Users can customize the MariSmart templates to generate specific MariSmart contracts. In future work, the customized MariSmart contracts will be verified with requirements extracted from domain knowledge. The source codes are publicly available for academic purposes at [17].

Our main contributions are summarized as follows:

- We extract the business logic model of maritime transportation from its domain knowledge.
- Based on the business logic model, we design the extendable MariSmart templates for accelerating the development of maritime transportation smart contracts.
- We carry out case studies on three real-world maritime transportation systems to show the feasibility and effectiveness of the MariSmart framework.

To our knowledge, we are the first to propose smart contracts from real-world trading agreements, which proves the feasibility of our framework for practical application.

The remainder of this paper is organized as follows. Section 2 introduces Ethereum and maritime transportation. We present the main work of our method in Sections 3 and 4. Section 5 presents case studies. Related work is reviewed in Section 7. Section 8 concludes and discusses future work.

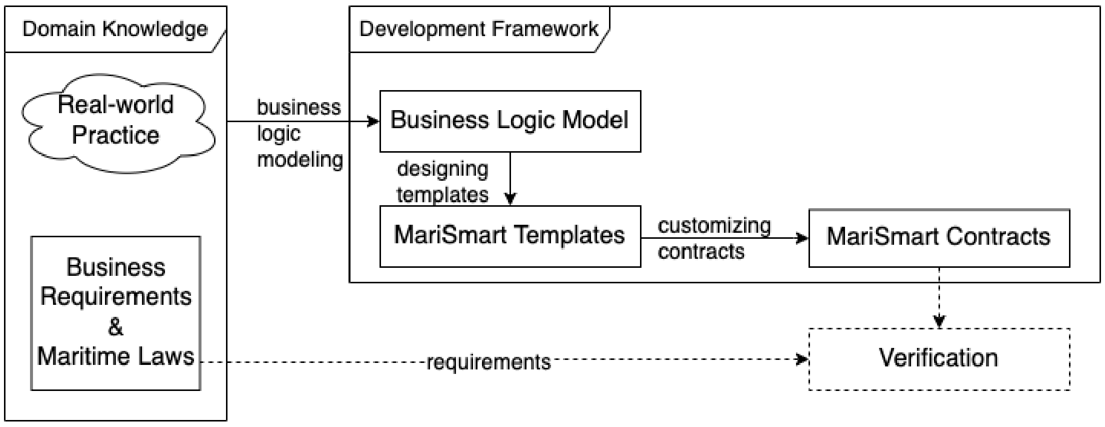


Figure 1. Overview of the MariSmart framework, the dash lined part will be proposed in future work

2. Preliminaries

2.1. Smart Contracts

Ethereum [2] is a blockchain, where the data and states are agreed upon by all participants. The states of the blockchain can be updated by *transactions*, which are initiated by *Externally-Owned Accounts (EOA)*. For complex interactions between EOA, Ethereum introduces smart contracts, which are codes that execute automatically under specific conditions. Both EOA and contract accounts can receive, hold, and send *Ethers*, and interact with deployed smart contracts.
Solidity is one of the most popular smart contract languages in Ethereum, which can be compiled and executed on the Ethereum Virtual Machine (EVM). In a *sol* source file, a smart contract definition contains declarations of State Variables, Functions, Function Modifiers, Events, Errors, Struct Types, and Enum Types. Especially, contracts can inherit from other contracts with the keyword *is*.
Take the smart contract in Figure 2 as an example, State Variables like *quantity* are permanently stored in contract storage, and their types and visibilities are defined in the declarations. In this case, the visibility is set to private by default. Functions like *exportShipment* are the execution units of the smart contract, and the conditions for calling it are restricted by Function Modifiers like *pre_exportShipment*, which usually contains *require* statements and add restrictions for calling the function. Events like *ShipmentExported* are defined for logging.
In Solidity, Function Modifiers are convenient for assigning the preconditions of the function. In a transaction where function *exportShipment* is called, the conditions in the modifier are checked first. The function can be executed only if the *require* statements are satisfied, otherwise, the whole transaction will be rolled back. As the function in a transaction may call other internal or external functions and form a call chain, all of the modifiers should be satisfied so that the transaction will take effect.
Inheritance is another important feature of Solidity. A smart contract can inherit from multiple contracts with keyword *is*,

```
contract Shipment is IShipment{
  uint quantity;
  uint weight;
  uint volume;
  uint price;
  uint down_payment;
  /* ... */
  modifier pre_exportShipment() virtual override
      {
    require(msg.sender == export_port_operator);
    require(state == State.inspected);
    _;
  }

  function exportShipment() external virtual
      override pre_exportShipment {
    state = State.exported;
    emit ShipmentExported(msg.sender, block.
        timestamp);
  }
  /* ... */
}
```

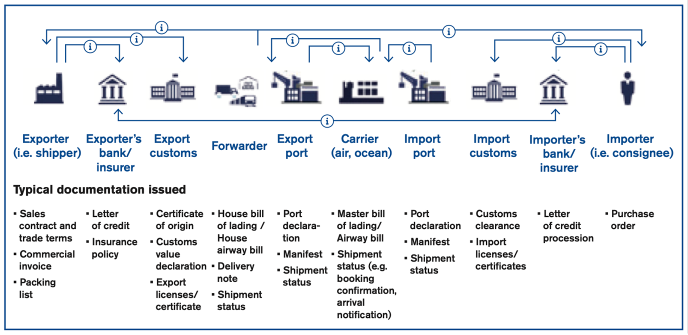
Figure 2. An example for Ethereum smart contracts.

where new functions, modifiers, and variables can be added (like the variable *quantity*), existing ones can be overridden (like the modifier *pre_exportShipment*), and the rest are inherited. In this way, codes can be reused conveniently, which reduces the coding work and improves the extensibility of the contracts.
The other keywords and features of Solidity will be explained when they are mentioned in the rest of the paper.

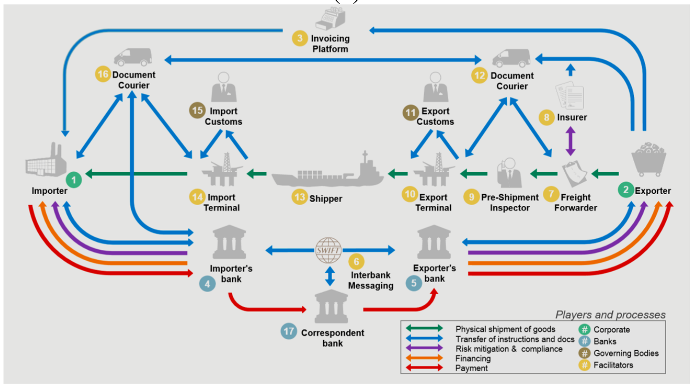
2.2. Maritime Transportation

Due to the low-cost and efficient features, maritime transportation plays a major role in the global supply chains, undertaking approximately 90% of the global trade [18]. Maritime transportation has suffered from burdensome paperwork. For example, a flower export consignment from Kenya to the Netherlands contains more than 200 bilateral communications among 20 organizations, and it usually takes about 10 days to finish the paperwork [13]. As is found in [19], a shipment can generate a pile of paper 25 cm high, and the cost of handling it can be higher than the cost of moving the containers. Smart contracts are thus a possible solution for paperless, transparent, and efficient transportation.

However, there is a lack of universal models for maritime transportation, which hinders the development of maritime smart contracts. Since there are multiple modes of maritime transportation, researchers usually focus on different aspects that vary from each other. For example, in the report [20] and [21] maritime transportation is modeled differently in the composition of participants and the workflow.



(a)



(b)

Figure 3. The business model of maritime transportation abstracted by [20] in (a), and by [21] in (b).

The diversity of the models result in inconsistency within the smart contracts for maritime transportation, which makes it hard to reuse the codes or extend the smart contracts. The stakeholders and main functions of [11] [12] [16] are listed in Table 1, although the main bodies of their business logic are similar, they ended up with different implementations of the stakeholders and workflows.

3. Business Logic Model

As we discussed in Section 2.2, it's necessary to provide a universal business logic model, before designing extendable templates. In this section, the business logic model of maritime transportation is modeled as a set of UML diagrams, depicting the stakeholders, activities, states of the shipment, and their relationships.

Table 1. Stakeholders and main functions of related work

Case	Stakeholders	Main Functions
[11]	container, sender_owner, receiver	CreatePackage, PerformmedSelfCheck, DepositMoneyforShipment, StartShippment, ShipmentArrived, ProvidePassphrase, UnlockShippment, GetShipmentMoney
[12]	agent, shipper, receiver, transporter	requestShipment, approveShipmentRequest, createNFT, issueBoL, claimCargo, cargoClaimDocumentsApproval, shipmentDelivered, burnContainerNFT
[16]	exporter, importer, freight-frw, agent, truck (transporter), sea (transporter)	requestShipment, documentsVerification, customsClearance, createUnimodalShipment, issueBoL, approveCarrierRequest, containerHandoff, documentsVerification, customsClearance, shipmentArrivedDestinationSignal

3.1. Stakeholders and Use Cases

Towards a general collection of stakeholders compatible with different models, we combine the stakeholders from [11] [12] [16] [20] [21] [22] [23] [24]. Some of them are replaced by the smart contract system, and the rest are abstracted into the following six types.

Shipper refers to the entity who owns the cargo, and is usually the seller. In real-world maritime transportation, a freight forwarder or forwarding agent is usually appointed by the owner, to arrange the carriage of goods on behalf of a shipper [22]. Since the freight forwarder interacts with other stakeholders in the same way as the shipper, we abstract the cargo owner, the freight forwarder as the shipper. The activities of the shipper include the following.

- Activity *create shipment* refers to the shipper filling out the information of shipment and creating the shipment.
- Activity *cancel shipment* refers to the shipper canceling the shipment before the carrier departs, who usually has to pay for half of the transportation fee.
- Activity *claim for compensation* refers to shipper claims for compensation if the shipment is damaged or lost.

Carrier refers to the entity that performs the transportation. In practice, the cargo may be transported in the form of multimodal transportation [25], where the transportation is segmented for several actual carriers. Besides, as implemented in [11], the smart container equipped with IoT sensors may also update the state of shipment as the carrier. To adapt to different scenarios, we abstract the contracting carrier, the actual carrier, and the smart container as the carrier. The activities of the carrier include the following.

- Activity *depart* refers to the carrier departing from the export port.
- Activity *report loss* refers to the carrier reporting the loss of shipment during transportation.
- Activity *arrive* refers to the carrier arriving at the import port.
- Activity *rearrange shipment* refers to the carrier resells or auctions the shipment if the consignee fails to receive the shipment within the agreed period.
- Activity *pay for compensation* refers to the carrier paying for the compensation if the shipment is damaged, delayed, or lost and compensation is claimed.

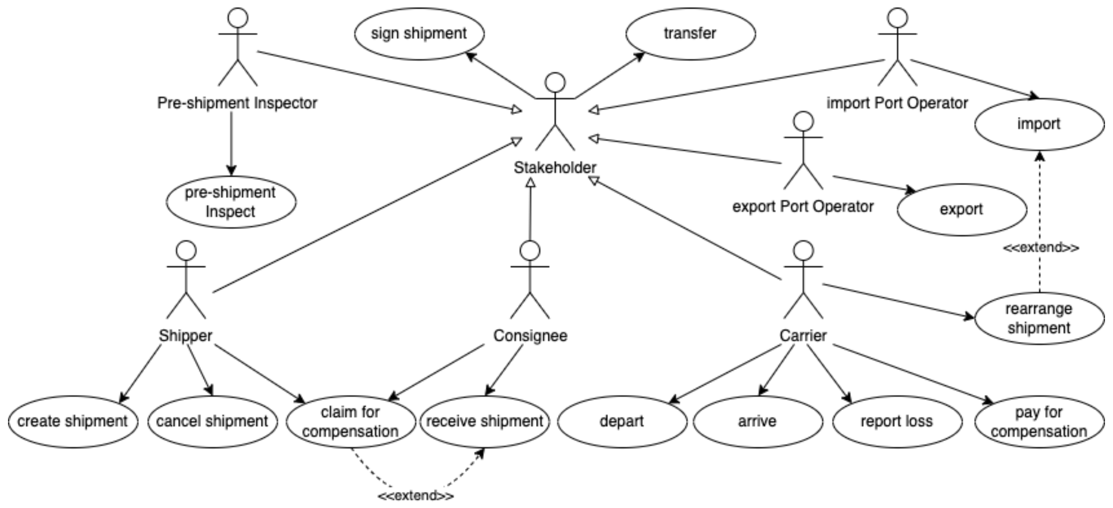


Figure 4. Use Case Diagram of maritime transportation systems

Consignee refers to the entity that receives the cargo and is usually the buyer. The activities of the consignee include activity *receive* and activity *claim for compensation*.

Pre-Shipment Inspector refers to the entity that inspects the shipment at embarkation ports or on the shipper's premises. The concept of activity *pre-shipment inspect* was first introduced by Zaire in 1963, and adopted by over fifty countries worldwide [23]. The pre-shipment inspector is usually independent from other participants and therefore included in our model. The pre-shipment inspector is mainly responsible for inspecting the shipment, where the status of the shipment will be checked.

Export/Import Port Operator refers to the entity that provides services at the export/import port. With the development of the port management theory, the services that a port can provide have been extended from simply loading and unloading to a wide range, including customs clearance, warehousing, etc. [24] Since a port is evolving to an aggregation of these entities, we abstract the customs, the customs agent, the warehouse owner, and the terminal operator as the export/import port operator, whose activity mainly includes activity *export/import*.

Other stakeholders such as the insurance companies, the banks, the document carriers, etc. are excluded from the model, since they do not directly participate the transportation, or can be replaced by smart contracts and blockchain systems.

The six stakeholders and their activities are abstracted and modeled as a Use Case Diagram. In Figure 4, each actor in the Use Case Diagram refers to a stakeholder, and each Use Case linked with an actor denotes its activity. All stakeholders inherit common Use Cases such as *sign the shipment*. The workflow of those Use Cases is modeled in the next section.

3.2. Workflow

Maritime transportation activities usually take place sequentially. However, the sequences and the conditions of the activities vary in different cases. Some activities are modeled specially so that it's compatible with different scenarios.

- Payment for Goods. In real-world maritime transportation, the time when the consignee pays the payment varies from before shipment, before release (like D/P and D/A), and after release (like O/A), etc. Therefore, we refactor the payment for goods in a down-payment way, where the consignee pays the down payment after the carrier departs, and pays the rest after receiving the shipment. Thus different modes of payment can be implemented by configuring the amount of the down payment and the rest.
- Compensation. As in by most of the domestic and international maritime law, the consignee and the shipper have the right to claim compensation when the shipment is delayed, damaged, or lost, yet only if he/she formally notices the carrier within a certain period after the date of arrival, and the compensation amount is limited. Considering the claimer and the amount of compensation varies in different situations, the compensation is modeled as the claimer first claim for compensation with an amount, then the carrier confirm and pay for it.

The workflow is modeled in an Activity Diagram, and additionally a State Machine Diagram. This is because the state of the shipment is implemented in most of the maritime smart contracts, yet some of which are error-prone. For instance, the function of reselling the shipment can be called under most circumstances in [12], which results in unexpected behaviors.

Activity Diagram contains activity nodes, control nodes, swimlanes, and edges, where the activity nodes basically come from the Use Case Diagram with amendments to cope with different scenarios, control nodes denote the start, end, and condition branches, swimlanes denote the stakeholders related to the activities, and edges denote the sequences of the activities, probably with guard conditions in the form '[guard]'. The Activity Diagram is shown in Figure 5.

State Machine Diagram consists of nodes and edges, where the nodes denote the states of the shipment, and the edges denote the transitions between the nodes, with the activities

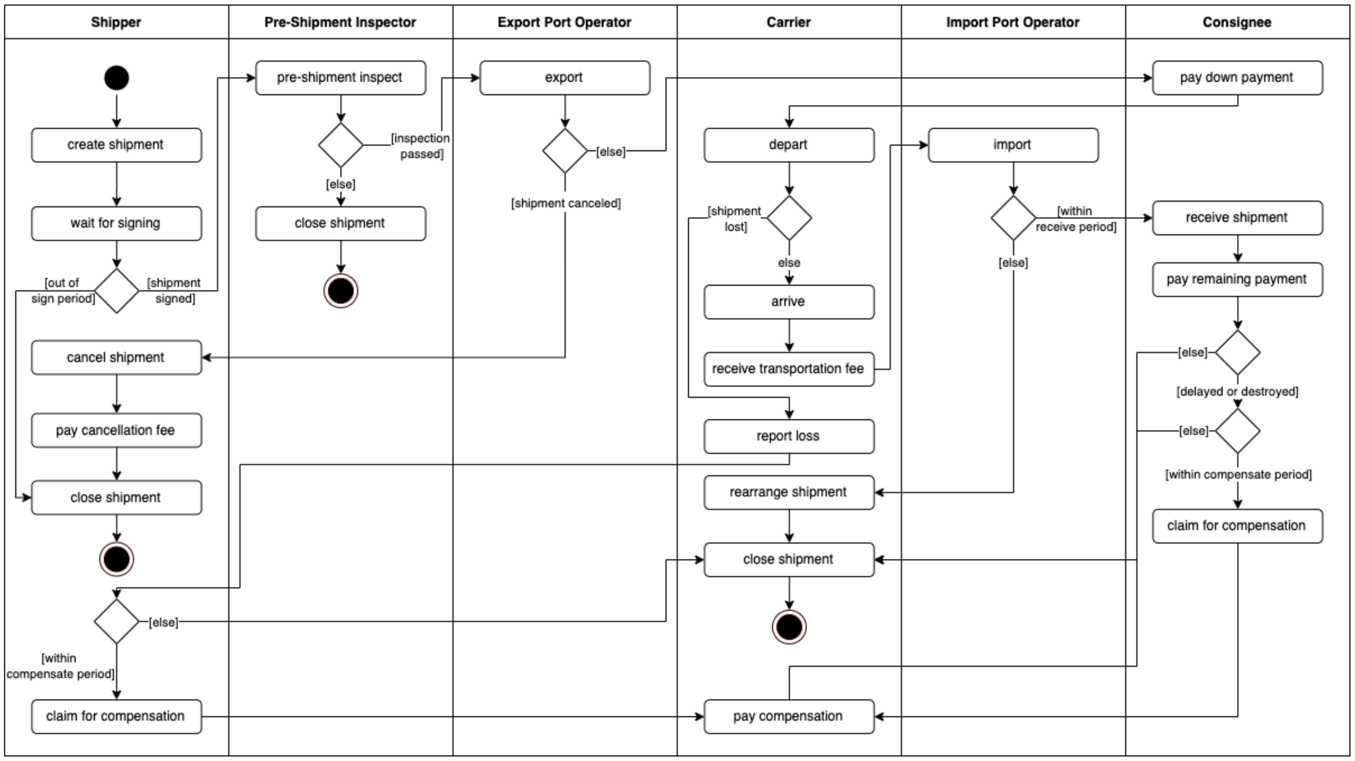


Figure 5. Activity Diagram of the workflow

as triggers and guard conditions in the form '[guard]'. The State Machine Diagram is shown in Figure 6.

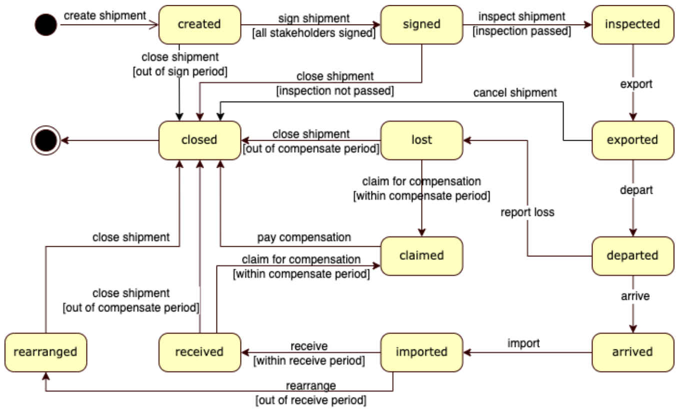


Figure 6. State Machine Diagram of the workflow

4. MariSmart Templates and Contracts

MariSmart Templates are a set of Solidity smart contracts, which can be extended into specific smart contracts for maritime transportation. Based on the business logic model proposed in Section 3, we propose the design of the MariSmart Templates, and how to customize them to MariSmart contracts.

4.1. Designing MariSmart Templates

The MariSmart templates consist of shipment templates and stakeholder templates, where the shipment templates maintain the state of shipment, and the stakeholder templates implement the activities of stakeholders.

The shipment templates consist of an interface *IShipment* and a contract *Shipment*. The elements of the shipment templates are defined in *IShipment* and implemented in *Shipment*, which contains variables, modifiers, and functions. The variables are used to maintain the state of the shipment and record the information of the shipment, including addresses, balances, and sign records of stakeholders, the parameters of the shipment, and the flags of the shipment like *is_lost*, *is_delayed* etc. The functions of the shipment template are used to update the state of the shipment and implement the necessary functionality, including transferring, emitting events, etc. The modifiers constrain the caller and the pre-condition of the functions with *require* statements.

The stakeholder templates are six contracts for specific stakeholders inherited from the contract *Stakeholder*. Each stakeholder template consists of a mapping from *UID* to *shipments* contract, and functions that implement the activities.

The following example explains the relationship between the MariSmart templates and the business logic model. The activity *cancel shipment* modeled in Figure 5 is implemented as the function *cancel* in Figure 7. The transition from *exported* to *closed* in Figure 6 is implemented in the function *cancelShipment* in Figure 7. When the shipper calls the function *cancel*, *cancelShipment* will be called and update the state of the shipment.

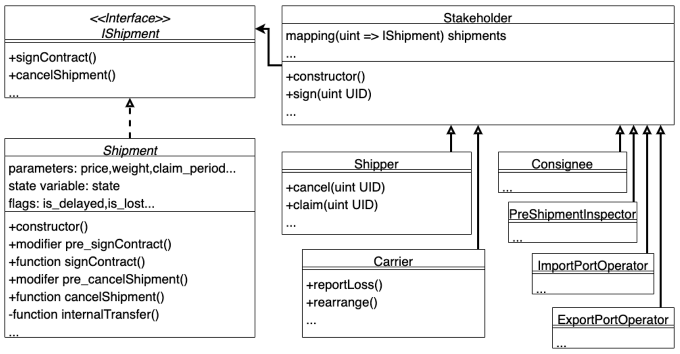


Figure 7. Relationships between MariSmart Templates

The important features of the MariSmart templates are implemented as follows.

Shipment Management. Each stakeholder template represents an entity or a firm and may be involved in multiple shipments at the same time. Therefore, the stakeholder template maintains a mapping from the shipment ID *UID* to the address of the corresponding shipment template. As for the shipper, this mapping is updated when the shipper creates the shipment. For other stakeholders, the address of a shipment template is recorded by the function *sign*. Once the shipment is created or signed, the stakeholder can assign the shipment ID as a parameter in further function calls.

Escrow balances. The payment between stakeholders is implemented by editing the mapping *balances* in the shipment template. The related variables and function are shown in Figure 8. When a stakeholder signs a shipment, a certain amount of Ethers required by *escrow_thresholds* are deposited to the shipment, and recorded in *balances*. When a payment occurs, instead of transferring directly, the function *internalTransfer* is called and *balances* are edited. When the shipment is closed, the balances can be withdrawn by the stakeholders.

Time issue. In Solidity, the keyword *block.timestamp* can be used to fetch the current timestamp of the transaction. Therefore, several time-related variables are defined in the shipment template, including *depart_date*, *arrive_date*, *receive_valid* etc., for comparing with *block.timestamp*, and *create_time*, *arrive_time* etc., for recording the timestamp at the corresponding event.

4.2. Customizing MariSmart Contracts

The MariSmart templates have a default configuration, which implements the basic function for maritime transportation. For specific business scenarios, the MariSmart templates can be customized according to the differences between the scenario and the business logic model.

According to the degree of customization, we divide smart contract development into three levels, namely customizing parameters, activities, and processes, which are proposed as

```
contract Shipment is IShipment{
  mapping(address => uint) balances;
  mapping(address => uint) escrow_thresholds;
  /* ... */
  function sign() external payable virtual override pre_sign {
    signatures[msg.sender] = true;
    balances[msg.sender] += msg.value;
    emit StakeholderSign(msg.sender, block.timestamp);
    if (
        signatures[shipper] == true &&
        signatures[carrier] == true &&
        signatures[consignee] == true &&
        signatures[pre_shipment_inspector] == true &&
        signatures[export_port_operator] == true &&
        signatures[import_port_operator] == true
    ) {
        state = State.signed;
        emit ShipmentSigned(msg.sender, block.timestamp);
    }
  }
  /* ... */
  function internalTransfer(
    address _from,
    address _to,
    uint _amount
  ) internal {
    require(balances[_from] >= _amount);
    require(_amount >= 0);
    balances[_from] -= _amount;
    balances[_to] += _amount;
    emit StakeholderTransfer(_from, _to, _amount, block.timestamp);
  }
}
```

Figure 8. Components of the shipment template related with escrow balances.

the following. The customization is mainly performed by overriding the functions and modifiers. In Solidity, the keyword *is* can be used to inherit smart contracts, and the keyword *override* is used for overriding.

Customizing parameters is the most basic way to customize MariSmart contracts, which adds and assigns parameters. For instance, the amount of down payment can be configured by setting the value of *down_payment* of the shipment template. Besides, the parameters of the stakeholder templates can also be configured.

Customizing activities is a more advanced way to customize MariSmart contracts and is used when the scenario requires a different implementation of an activity. Such customization is performed by the keyword *override*. For example in Figure 9, the function of *depart* of the carrier template can be rewritten to update the ownership of the cargo additionally.

Customizing workflow is used when the scenario requires a different workflow, such as different transitions of the shipment state, different composition of stakeholders, etc. It is performed by overriding the modifiers of the shipment template. Note that this type of customization may result in

```
contract Carrier is Stakeholder {
  /* codes above */
  function depart(uint _UID) public virtual
      onlyOwner {
    shipments[_UID].depart();
  }
  /* codes below */
}
```

(a)

```
contract NFTCarrier is Carrier {
  /* codes above */
  function depart(uint _UID) public override
      onlyOwner {
    shipments[_UID].depart();
    /* custom logic here */
    ContainerNFT(ContainerNFT_addr).transferFrom(
      shipments[_UID].getShipper(),
      address(this),
      NFTShipment(address(shipments[_UID])).
          getNFTID()
    );
  }
/* codes below */
}
```

(b)

Figure 9. Customizing the depart activity (b) by overriding the templates (a)

unexpected behavior, and should be carefully considered.

Theoretically, the MariSmart templates can be customized arbitrarily. However, for the security and stability of the MariSmart contracts, the following recommendations are proposed.

- It's strongly recommended to avoid transferring Ethers directly. Use *externalTransfer* instead, which edits the escrow balances. (please refer to Section 6.1 for detail)
- When the workflow is customized, it's recommended to make sure that the sequences of function calls cannot be reversed, otherwise, the MariSmart contracts may be transactions order dependant. (please refer to Section 6.1 for detail)

5. Case Studies

In this section, we carry out case studies on three real-world systems, using the MariSmart framework to rewrite the former two and code the third one. We first analyze the difference between the particular case and the business logic model proposed in Section 3, then customize the MariSmart contracts using the method in Section 4.2.

5.1. An IoT-based Smart Contract for Tracing and Monitoring Cargo Containers

Hasan et al. [11] proposed a Solidity smart contract to monitor shipment conditions through IoT sensors. A shipper, consignee, and container are included in the smart contract. As is discussed in Section 3, a container can be abstracted as a carrier. Thus the stakeholders are consistent with the business logic model.

Compared with the business logic model, three features can be extracted from the scenario.

- When the consignee pays for the goods, a hash code is generated to authorize the consignee to access the container.
- If exceptions occur during transportation, the corresponding event will be triggered by the container, the shipment will be canceled and all payments will be refunded.
- When the consignee receives the shipment, the container has to provide the correct hash code within 48 hours. Otherwise, the shipment will be canceled and half of the payment will be refunded.

The first feature can be implemented by adding a new attribute *passcode_hashes* to the consignee contract, and overriding its *sign* function so that the consignee updates the hash code while signing the shipment. The second feature can be implemented by overriding the function *reportLoss* of the carrier contract, and adding corresponding events and data type to it. The third feature can be implemented by overriding the function *receiveShipment* of the consignee contract.

5.2. A NFT Based Smart Contract for Tracing and Auctioning Shipping Cargo

Elmay et al. [12] proposed a smart contract to trace the ownership change and auction the cargo. Four stakeholders are involved in the proposed smart contract, namely a shipper, a ship line agent, a transporter, and a consignee. As is discussed in Section 3, the ship line agent and the actual carrier can be abstracted as the carrier in the business logic model.

Compared with the business logic model, differences can be extracted from the scenario.

- A NFT is minted by the NFT contract when the shipment is created, and the ownership of the NFT is transferred as the ownership of the shipment. When the consignee receives the shipment, or the shipment is auctioned, the NFT is burned.
- If the consignee does not receive the shipment after its arrival, the shipment will be auctioned through an auction contract.

The first feature can be implemented by importing the NFT contract, and adding relevant statements to the contracts of shipper, carrier, and consignee, including *create*, *cancel*, *depart*, *arrive*, *rearrange*, *close*, *receiveShipment*. As for the auctions, the auction contract implemented in the proposed smart contract is imported and the function *rearrange* and *close* of the carrier contract are added with relevant statements to start and close the auction.

Notably, in the original implementation [12], the function *auctionCargo* can be called at any time except when the consignee claims for the shipment or the agent approves the claim, which may result in unexpected behaviors. By refactoring as MariSmart contracts, the state of the shipment is maintained correctly in the customized smart contracts.

5.3. A LNG Sales Agreement

The master ex-ship LNG sales agreement [26] was signed by Cheniere Marketing, Inc. and Gaz de France International Trading S.A.S. in 2007, and made public by the U.S. Securities and Exchange Commission (SEC). The agreement contains the basic information about the shipment, the payment, the risk and liability, the claims and disputes, etc. Due to the nature of LNG transportation, the quality of LNG is examined at the import port, specifically when unloading from the vessel. Therefore, the pre-shipment inspection is not included in this scenario.

The process of transportation in the agreement varies from the default implementation as follows.

- According to Letter 7, the shipper has the right to cancel the shipment before the carrier leaves the export port, but has to pay a cancellation fee to the consignee.
- According to Letter 7.1.1, the carrier has to notify the consignee at a specific time before arrival
- According to Letter 4.3, the carrier has the right to leave the port if the consignee fails to receive the shipment within 48 hours after arrival. In this case, the shipper is permitted to claim compensation from the consignee. However, if the shipper gains profit from the resale, the shipper has to return the profit to the consignee, no more than the value of goods.

Accordingly, the smart contract generated from the templates is customized as follows.

- Due to the absence of the pre-shipment inspector, the state transitions vary from the State Machine Diagram. Thus the relevant modifiers of the shipment contract, namely *pre_inspect* and *pre_exportShipment* are overrided.
- The function *cancel* of the shipper contract is overridden to implement the cancellation fee.
- The function *arrive* of the carrier contract is overridden to implement the notification of arrival. Besides, the function *notify* is added to the carrier contract to emit the corresponding event.
- The function *rearrange* of the carrier contract is overridden to implement the resale as required.

6. Discussion

6.1. Security Analysis

Re-entrancy is a typical smart contract vulnerability. When Ethers are transferred by the function *send* or *transfer*, a malicious receiver can alter its *fallback* function and gain more Ethers than they ought to be. There are two techniques applied in MariSmart contracts to avoid re-entrancy. Firstly, due to the escrow system, stakeholders do not actually *send* or *transfer* Ethers to each other, and the escrow balances in the shipment contract are edited instead. No fallback functions are possibly called and therefore most of the functions in MariSmart contracts are free from re-entrancy. Secondly, there is only one function in the shipment contract that contains a transfer statement, namely *withdraw* in Figure 10. The *withdraw* function first sets the balances to zero, then calls function *transfer*. The *fallback* function cannot trigger *withdraw* again, since the modifier is not satisfied anymore.

Transaction Order Dependancy is another common vulnerability. When the results of two transactions depend on the order of the transactions, a malicious miner can control the result by assigning the order. As for MariSmart contracts, there is either only one function that can be possibly called, or multiple

```
modifier pre_withdraw() virtual override {
    require(state == State.closed);
    require(balances[msg.sender] > 0);
    _;
}
function withdraw() external virtual override
    pre_withdraw {
    uint amount = balances[msg.sender];
    balances[msg.sender] = 0;
    payable(msg.sender).transfer(amount);
    emit StakeholderWithdraw(msg.sender, amount,
        block.timestamp);
  }
```

Figure 10. The only one function of the shipment contract that transfers Ethers

functions are possible, but they cannot take place sequentially. Therefore, the sequences of transactions cannot be reversed and the MariSmart contracts are free from transaction order dependency.

Accessability is an important issue in maritime transportation since multiple stakeholders are involved and each of them is entitled to different variables and functions. The accessibility in MariSmart contracts is designed from two aspects. Firstly, the variables of the shipment contract are set to be private, and the stakeholders access the shipment variables through getter functions. This way, the variables are free from being edited by external addresses, and the accessibility can be conveniently assigned by the modifiers of the getter functions. Secondly, the accessibility of the functions is maintained by modifiers.

6.2. Comparision with Other Work

Functions and Features supported by MariSmart and related work are compared in Table 2, where *MariSmart* stands for the default configuration of the MariSmart contracts. The implementation of these features is discussed in Section 3 and 4. Notably, the compensation is partly implemented in [11] because compensation for canceling shipment is included in the contract, yet compensation for losing or destroying shipment is not implemented. Similarly, the smart contracts of [16] are partly extendable because there are alternative carriers, namely sea carriers, inland carriers, and air carriers, but other stakeholders and activities are fixed.

Efficiency improvement is evaluated by reduction and reuse of the codes. In Table 3, *Source* stands for the lines of the original implementation, *Customization* stands for the lines that developers have to code to customize MariSmart contracts, *Total* stands for the total lines of MariSmart contracts, including the templates and interfaces.

Although we rewrite the [12] and [11] with more stakeholders and activities, the necessary codes are reduced by 49.63% and 43.62% compared with their original versions. Since the backbone of the MariSmart contracts are implemented in the MariSmart templates, these reused codes account for from 75.90% to 89.06% of the total lines.

Table 2. Supported features of related work

Case	payment	cancellation	logging	compensation	rearrangement	extendable
[12]	✗	✗	✓	✗	✓	✗
[16]	✗	✗	✓	✗	✗	partly
[11]	✓	✓	✓	partly	✗	✗
MariSmart	✓	✓	✓	✓	✓	✓

Table 3. Codes statistic

Case	Source(lines)	Customization(lines)	Reduced(%)	Total(lines)	Reused(%)
NFT	544	274	49.63%	1137	75.90%
IoT	188	106	43.62%	969	89.06%
LNG	/	116	/	979	88.15%

7. Related Work

Our work is mainly related to two kinds of work, namely smart contract application in maritime transportation and development through domain-specific language.

Smart contract applications are partly discussed in Section 5, where the smart contracts from [11] and [12] are rewritten as MariSmart contracts. In the following work of Elmay et al. [16], the business model is expanded and compatible with single-mode and multi-mode transportation. In Section 5, we argue that the MariSmart templates can be extended to such scenarios, and from Section 6, we argue that the customized MariSmart contracts are more functional than the original implementation.

Domain-specific languages are proposed to develop smart contracts understandably and efficiently. Skotnica et al. [27] proposed DasContract. The following work [28] [29] simplified the model and implemented an automatic translation. Hamdaqa et al. [30] proposed iContractML, and the 2.0 version [31] added support for DAML. Wöhrer et al. [32] proposed CML based on Xtext [33]. It is worth noting that [32] and [30] only generate the skeleton of smart contracts, and users still need to implement the activities. As for DasContract, the code can be automatically generated, yet developers still have to implement the business logic in the intermediate language. From this aspect, the domain-specific languages and the Marismart framework improve efficiency in different ways, and are not contradictory. MariSmart framework could introduce intermediate languages and code generation techniques in future work.

8. Conclusion

In this paper, we have proposed a development framework, MariSmart, to develop maritime transportation smart contracts. The MariSmart framework is flexible for different maritime transportation scenarios and can be extended for more complex business modes. Three case studies have been carried out to show the feasibility and effectiveness of our work.

For future work, as we mentioned in Section 1, a verification framework will be proposed to improve the reliability of MariSmart contracts. The framework will model the MariSmart contracts and the requirements extracted from domain knowledge, and automatically finish the verification. In MariSmart contracts, several features can bring convenience for future verification. Firstly, there are variables like *depart_time* and *arrive_time*, which record the time when the corresponding activities take place so that the important events can be traced and modeled easily. Secondly, all activities are implemented as functions of the same name in a similar structure, which makes it easy to establish a mapping between the smart contract components and the activities. Lastly, since the MariSmart contracts are developed by inheriting the templates, only the incremental part is to be dealt with, which reduces the difficulty of verification.

References

[1] N. Szabo, "Smart contracts.[online]," http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html, 1994.

[2] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, pp. 2–1, 2014.

[3] F. Schär, "Decentralized finance: On blockchain-and smart contract-based financial markets," *FRB of St. Louis Review*, 2021.

[4] H. Subramanian, "Security tokens: architecture, smart contract applications and illustrations using safe," *Managerial Finance*, vol. 46, no. 6, pp. 735–748, 2020.

[5] A. Khatoon, "A blockchain-based smart contract system for healthcare management," *Electronics*, vol. 9, no. 1, p. 94, 2020.

[6] A. Saini, Q. Zhu, N. Singh, Y. Xiang, L. Gao, and Y. Zhang, "A smart-contract-based access control framework for cloud smart healthcare system," *IEEE Internet of Things Journal*, vol. 8, no. 7, pp. 5914–5925, 2020.

[7] K. N. Griggs, O. Ossipova, C. P. Kohlios, A. N. Baccarini, E. A. Howson, and T. Hayajneh, "Healthcare blockchain system using smart contracts for secure automated remote patient monitoring," *Journal of medical systems*, vol. 42, pp. 1–7, 2018.

[8] S. Wang, D. Li, Y. Zhang, and J. Chen, "Smart contract-based product traceability system in the supply chain

scenario," *IEEE Access*, vol. 7, pp. 115 122–115 133, 2019.

[9] L. Wang, L. Xu, Z. Zheng, S. Liu, X. Li, L. Cao, J. Li, and C. Sun, "Smart contract-based agricultural food supply chain traceability," *IEEE Access*, vol. 9, pp. 9296–9307, 2021.

[10] V. Astarita, V. P. Giofrè, G. Mirabelli, and V. Solina, "A review of blockchain-based systems in transportation," *Information*, vol. 11, no. 1, p. 21, 2019.

[11] H. Hasan, E. AlHadhrami, A. AlDhaheri, K. Salah, and R. Jayaraman, "Smart contract-based approach for efficient shipment management," *Computers & industrial engineering*, vol. 136, pp. 149–159, 2019.

[12] F. K. Elmay, K. Salah, R. Jayaraman, and I. A. Omar, "Using nfts and blockchain for traceability and auctioning of shipping containers and cargo in maritime industry," *IEEE Access*, vol. 10, pp. 124 507–124 522, 2022.

[13] G. Balci and E. Surucu-Balci, "Blockchain adoption in the maritime supply chain: Examining barriers and salient stakeholders in containerized international trade," *Transportation Research Part E: Logistics and Transportation Review*, vol. 156, p. 102539, 2021.

[14] P. Dutta, T.-M. Choi, S. Somani, and R. Butala, "Blockchain technology in supply chain operations: Applications, challenges and research opportunities," *Transportation research part e: Logistics and transportation review*, vol. 142, p. 102067, 2020.

[15] S. Nguyen, P. S.-L. Chen, and Y. Du, "Risk assessment of maritime container shipping blockchain-integrated systems: An analysis of multi-event scenarios," *Transportation Research Part E: Logistics and Transportation Review*, vol. 163, p. 102764, 2022.

[16] F. K. Elmay, K. Salah, I. Yaqoob, R. Jayaraman, A. Battah, and Y. Maleh, "Blockchain-based traceability for shipping containers in unimodal and multimodal logistics," *IEEE Access*, vol. 10, pp. 133 539–133 556, 2022.

[17] "Marismart framework. [online]," https://github.com/MariSmartSourceCode/MariSmart, 2023.

[18] D. Song, "A literature review, container shipping supply chain: Planning problems and research opportunities," *Logistics*, vol. 5, no. 2, p. 41, 2021.

[19] I. Allison, "Shipping giant maersk tests blockchain-powered bill of lading," *International Business Times*, vol. 14, 2016.

[20] E. Ganne, "Can blockchain revolutionize international trade?[online]," https://www.wto.org/english/res_e/booksp_e/blockchainrev18_e.pdf, 2018.

[21] B. C. Group, "Digital innovation in trade finance: Have we reached a tipping point?[online]," https://www.swift.com/news-events/news/digital-innovation-trade-finance-have-we-reached-tipping-point, 2017.

[22] K.-C. Shang and C.-S. Lu, "Customer relationship management and firm performance: an empirical study of freight forwarder services," *Journal of Marine Science and Technology*, vol. 20, no. 1, p. 8, 2012.

[23] J. Anson, O. Cadot, and M. Olarreaga, "Tariff evasion and customs corruption: does pre-shipment inspection help?" *The BE Journal of Economic Analysis & Policy*, vol. 5, no. 1, p. 0000101515153806451600, 2006.

[24] D. Olivier and B. Slack, "Rethinking the port," *Environment and Planning A*, vol. 38, no. 8, pp. 1409–1427, 2006.

[25] M. SteadieSeifi, N. P. Dellaert, W. Nuijten, T. Van Woensel, and R. Raoufi, "Multimodal freight transportation planning: A literature review," *European journal of operational research*, vol. 233, no. 1, pp. 1–15, 2014.

[26] M. Keith and S. Edward, "Master ex-ship lng sales agreement between cheniere marketing, inc. and gaz de france international trading s.a.s." https://www.sec.gov/Archives/edgar/data/3570/000119312507106384/dex102.html, 2007.

[27] M. Skotnica and R. Pergl, "Das contract-a visual domain specific language for modeling blockchain smart contracts," in *Enterprise Engineering Working Conference*. Springer, 2019, pp. 149–166.

[28] B. Hornáčková, M. Skotnica, and R. Pergl, "Exploring a role of blockchain smart contracts in enterprise engineering," in *Advances in Enterprise Engineering XII: 8th Enterprise Engineering Working Conference, EEWC 2018, Luxembourg, Luxembourg, May 28–June 1, 2018, Proceedings 8*. Springer, 2019, pp. 113–127.

[29] M. Skotnica, J. Klicpera, and R. Pergl, "Towards model-driven smart contract systems–code generation and improving expressivity of smart contract modeling," *Proc. EEWC*, vol. 20, 2020.

[30] M. Hamdaqa, L. A. P. Metz, and I. Qasse, "Icontractml: A domain-specific language for modeling and deploying smart contracts onto multiple blockchain platforms," in *Proceedings of the 12th System Analysis and Modelling Conference*, 2020, pp. 34–43.

[31] M. Hamdaqa, L. A. P. Met, and I. Qasse, "icontractml 2.0: A domain-specific language for modeling and deploying smart contracts onto multiple blockchain platforms," *Information and Software Technology*, vol. 144, p. 106762, 2022.

[32] M. Wöhrer and U. Zdun, "Domain specific language for smart contract development," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2020, pp. 1–9.

[33] M. Eysholdt and H. Behrens, "Xtext: implement your language faster than the quick and dirty way," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, 2010, pp. 307–309.