Formalising a Gateway-based Blockchain Interoperability Solution with Event-B

Author 1
Affiliation
Organization
City, Country
email

Author 2
Affiliation
Organization
City, Country
email

Author 3
Affiliation
Organization
City, Country
email

***Abstract*—In recent years, multiple solutions have been proposed for blockchain interoperability. However, designing these solutions is complex, and design failures have caused great economic damage to their owners. Safety and liveness are essential properties for these solutions, and formalisation eases their verification. However, only a few efforts were performed to formalise interoperability solutions with known formal methods. TLA+ and PAT were previously used; however, the Event-B method has not been explored, although it might be a suitable approach. The purpose of this paper was to explore the formalisation of a gateway-based interoperability solution with Event-B. The results showed that the method was suitable and that a straightforward specification could be developed considering Ethereum and Hyperledger Fabric as the involved blockchains. The specification was assessed with three strategies that enabled its verification and validation. In particular, formal verification (e.g. safety properties), functional validation, and functional utility. These promising results constitute a step forward in the development of formal specifications for blockchain interoperability solutions.**

***Index Terms*—distributed ledger technology, blockchain, interoperability, event-b, ethereum, hyperledger fabric**

I. Introduction

Blockchain interoperability has been one of the main research topics surrounding blockchain in recent years [1]–[3]. Multiple approaches have been proposed to support it [1] and the Gateway-based approach is one of them [4]. This approach comprises pieces of software hosted aside each blockchain that know how to interact with it (i.e. submit transactions and listen to events). In addition, gateways define a protocol to exchange messages between them, enabling blockchain interoperability by exchanging cross-chain transactions. This approach was followed by several authors [5]–[8]. However, developing them is costly and complex, as designers must ensure that their design decisions guarantees the required safety and liveness properties [9]. Safety properties prevent an undesirable state or event from occurring, whereas the liveness properties imply that a state or event will eventually occur in the system [10]. Formalising blockchain interoperability solutions is still a challenge [11]. Indeed, current gateway-based interoperability solutions do not provide a formal specification, hindering the verification of such properties.

Recent studies have formalised specific interoperability solutions using the TLA+ and PAT methods [12]–[14], although the gateway-based approach has not yet been formalised. Furthermore, the Event-B method has not been used to formalise interoperability solutions, although it enables verification of safety properties and verification of functional correctness of a system according to a specification [15]. In addition, the Event-B method provides capabilities to verify smart contract correctness and safety properties [16], [17], which makes it suitable for the gateway-based approach, as it relies on smart contracts. Taking this into account, this study selected the gateway proposed by Pandolfi et al. [8] as a case study to evaluate the suitability of the Event-B method to formalise the gateway-based approach and give insights to answer the following research question: **How can the gateway-based approach be formalised with Event-B?**

The results showed that it was possible to formalise a specific gateway solution with Event-B. Furthermore, the method was suitable, as the model obtained was straightforward. It was not necessary to push its limits to comply with the requirements. The method enabled the specification of the gateway using different levels of abstraction, facilitating its extensibility (e.g. incorporating blockchains). Three evaluation strategies were used to assess the specification. Proofs were generated for each model, which proved its functional correctness and the verification of safety properties. Animations were generated to validate the functional behaviour of the specification, and a use case scenario was developed to validate its functional usage.

The main contributions of this paper are: 1) the confirmation that it was possible to formalise a specific gateway solution with Event-B, verifying its safety properties. This positions Event-B as a candidate method to formalise the gateway-based approach; 2) a straightforward formal specification that is extensible to support multiple blockchains that unambiguously describes the gateway behaviour; and 3) evaluation strategies (i.e. formal verification, functional validation, and functional utility of the specification) that enabled us to evaluate the proposal and may guide future studies to apply Event-B to model other interoperability solutions.

The remainder of the paper is organised as follows. Section II provides background concepts. Section III describes the Event-B model of the gateway. Section IV presents how the model was evaluated as well as a discussion of the results and its limitations. Section V analyses the related work. Finally, Section VI presents conclusions and future work.

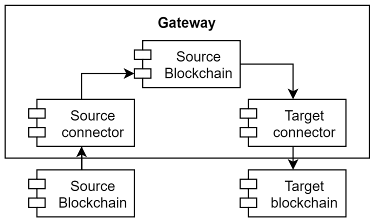

Fig. 1: Gateway-based interoperability solution

II. Background

This section presents the background concepts.

A. Gateway-based approach for Blockchain Interoperability

Pandolfi et al. [8] proposed a general-purpose blockchain interoperability solution following the gateway approach, to enable interoperability between Ethereum [18] and Fabric [19]. This solution takes advantage of the smart contract and events capabilities of each blockchain. Fig. 1 presents a high-level overview of the proposal. For each blockchain, there exists a connector that can interact with it. Whenever a smart contract on the source blockchain triggers a cross-chain event, this event is listened by the blockchain connector. The connector transforms the received event into a canonical message data format and redirects the message to the router component. Each message has a blockchain and a smart contract destination that enable the Router component to route it to the connector related to the destination (i.e. target connector). The target connector receives the canonical message, transforms it into the target blockchain data format, and submits a transaction to the target smart contract. The gateway was tested with Ethereum and Fabric for one-way and request-response cross-chain smart contract invocation.

B. Event-B

Event-B is a method based on first-order logic and type-based theory that can be used to formalise systems that can be modelled as discrete transition systems [20]. Event-B models enable the verification of the functional correctness of the system and the safety properties they must comply with.

Event-B models consist of static (i.e. Context) and dynamic components (i.e. Machine). The Context may contain carrier sets, axioms, and constants. The sets are user-defined types, constants that represent the values that variables of a Machine may have, and axioms are the presumed properties of carrier sets and constants. A Machine may have variables that represent its state, invariants to constraint the state of these variables, and events that represent its dynamic behaviour. Invariants must be proved to hold in every change of the Machine's state. Events may contain input parameters, guards, and actions. Event's guards represent the preconditions needed for its execution. The preconditions must be met before the event can be executed. An event's actions may change the state of the Machine after the event is executed.

Refinement is one of the main capabilities of Event-B to deal with the complexities of modelling a system. A machine can be refined to a new machine with new details of the model

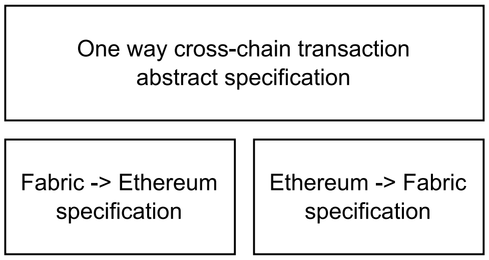
One way cross-chain transaction abstract specification
Fabric -> Ethereum specification
Ethereum -> Fabric specification

Fig. 2: Specification layers

(e.g. new events). The refined machine is called an abstract machine, and the refinement is called a concrete machine.

Event-B defines proof obligations (POs) that specify what needs to be proved in the model. They are properties that must be verified to demonstrate that the model is sound and respects its behavioural semantics. Some examples of POs are invariants preservation and well-definedness of the model.

Finally, Rodin is an open source platform to aid in the construction and verification of Event-B models [21]. Rodin can be used with a ProB model checker and animator to validate and animate the model. Rodin comes with a set of provers that can automatically discharge POs.

III. Event-B Specification

This Section presents the Event-B specification of the gateway-based solution presented in Section II-A. The specification enables a formal verification of the functional correctnes and the required safety properties of the gateway. The Event-B method is appropriate for modelling the gateway-based solution, as the gateway can be modelled as a discrete transition system. Furthermore, the incremental modelling of Event-B helps to conceptualise the gateway, from an abstract specification such as source/target blockchain, smart contract, and gateway, to subsequent model refinements that introduce details of the specific blockchains (e.g. Ethereum). The abstract specification expresses the behaviour of the gateway independently of the blockchains used. As described later in Section III-A, this behaviour is limited and is concerned with smart contracts only. Refinements include the specific behaviour introduced by the selected blockchains that need to interoperate. For example, Fabric requires changes on the behaviour of the abstract system, as all submitted transactions must be authenticated and authorised. On the other hand, Ethereum requires users to pay for each transaction submitted, which changes the behaviour of the gateway.

Fig. 2 presents the specification through its different layers. The first layer, called an abstract specification, formally defines the architecture and behaviour of the gateway without the details of a specific blockchain. This layer is refined in two new layers that extend its behaviour, adding the details required to interoperate Ethereum and Fabric. One layer specifies the behaviour of one-way smart contract invocation from Fabric to Ethereum, and the other layer from Ethereum to Fabric.

A. Abstract specification

The first layer of the specification models the relevant features that are independent of the specific blockchains (e.g.

Ethereum and Fabric). This way, the three components of the gateway (i.e. router and connectors) were abstracted into a single component (i.e. gateway). The source and target blockchains were omitted, and only the source and target smart contracts were included. Furthermore, communication protocols were omitted, and transactions were simplified to an abstract concept without modelling specific business data.

Fig. 3 illustrates the behaviour of the modelled gateway. First, the gateway subscribes to the events triggered by the smart contract on the source blockchain. Second, the user initiates a cross-chain transaction (cc-tx). Third, the source smart contract processes the cc-tx and triggers a cc-tx event. Fourth, the gateway listens to the cc-tx event and submits a cc-tx to the target smart contract on the target blockchain. This behaviour is described in Table I through four requirements.

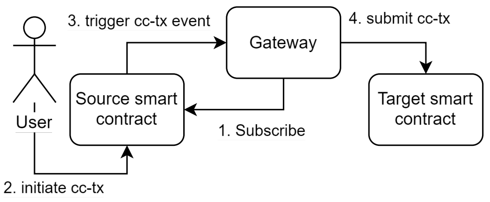

Fig. 3: Abstract gateway behaviour

TABLE I: Abstract specification requirements

ID	Description
RQ1	The Gateway can subscribe to the source smart contract events.
RQ2	A user can initiate a cc-tx on a smart contract on the source blockchain.
RQ3	The source smart contract triggers an event for each submitted transaction. This event becomes a cc-tx to a target blockchain.
RQ4	When the gateway receives an event from the source smart contract, then submits a cc-tx to the target smart contract with the received data.

Fig. 4 presents the context of the abstract machine. The gateway, source, and target smart contracts were modelled through Event-B constants. All these constants have a type that was modelled as Event-B sets. Some sets were cross-chain transactions, and cross-chain smart contracts, among others.

```
sets GATEWAYS TRANSACTIONS
  CROSS_CHAIN_SMART_CONTRACTS
  CROSS_CHAIN_EVENTS CROSS_CHAIN_TRANSACTIONS

constants source_smart_contract target_smart_contract
  gateway

axioms
 @axm1 source_smart_contract ∈
  CROSS_CHAIN_SMART_CONTRACTS
 @axm2 target_smart_contract ∈
  CROSS_CHAIN_SMART_CONTRACTS
 @axm3 gateway ∈ GATEWAYS
```

Fig. 4: Context of the abstract machine

Fig. 5 shows how the gateway can subscribe to the source smart contract events. This requirement (RQ1) is modelled using an Event-B event with only one guard. This guard checks if the gateway does not already have a subscription to the

```
invariants
  @inv3 subscriptions ∈ GATEWAYS ↔
    CROSS_CHAIN_SMART_CONTRACTS

event SUBSCRIBE_SMART_CONTRACT_EVENTS
  where
   @grd1 gateway ↦ source_smart_contract ∉ subscriptions // The
    gateway is not already subscribed to the smart contract events
  then
   @act1 subscriptions := subscriptions ∪ {gateway ↦
    source_smart_contract} // The gateway is subscribed to listen to
    the smart contract events
 end
```

Fig. 5: Subscribe to cross-chain smart contract events (RQ1)

```
invariants
 @inv1: received_transactions ∈
    CROSS_CHAIN_SMART_CONTRACTS ↔ TRANSACTIONS

event INITIATE_CC_TX
  any transaction
  where
   @grd1 transaction ∈ TRANSACTIONS
   @grd3 transaction ∉ received_transactions[{
    source_smart_contract}] // The transaction was not received by the
     smart contract
  then
   @act1 received_transactions := received_transactions ∪ {
    source_smart_contract ↦ transaction} // Add the transaction to
    the received transactions of the smart contract
end
```

Fig. 6: Submit cross-chain transaction specification (RQ2)

smart contract. After the event is executed, a new subscription is created. Subscriptions are modelled as a Cartesian product of gateways and cross-chain smart contracts (see invariant).

Fig. 6 presents how the second requirement (RQ2) was modelled, where a user initiates a cc-tx on the source smart contract. This requirement was also modelled as a machine event with two guards. The most relevant guard is that the transaction was not already submitted to the smart contract. After the event execution, the transaction is added to the received transactions of the smart contract. These transactions are pending to be processed by the smart contract.

Fig. 7 presents how requirement three (RQ3) was modelled. This event has two guards: 1) the smart contract has a pending transaction to process, and 2) the smart contract will always trigger a new cc-tx event. The execution of this event creates a new relationship between the new cc-tx event and the source smart contract (i.e. the smart contract triggers a cc-tx event). Furthermore, the transaction is removed from the pending transactions to be processed by the source smart contract.

Requirement four (RQ4) was modelled with two events: listen to the cc-tx event and submit the cc-tx. The first event represents the gateway listening to the events triggered by the source smart contract (see Fig. 8). This event can only be executed if 1) the source smart contracts have triggered a cc-tx event, 2) the gateway has a subscription to the source smart contract, and 3) the gateway has not already listened to the cc-tx event. After the event is executed, the cc-tx event

```
invariants
  @inv2; triggered_events ∈
    CROSS_CHAIN_SMART_CONTRACTS ↔
    CROSS_CHAIN_EVENTS

 event TRIGGER_CC_TX_EVENT
  any transaction cross_chain_event
  where
   @grd1 source_smart_contract ↦ transaction ∈
     received_transactions // The smart contract has a pending
     transaction to process
   @grd2 cross_chain_event ∉ triggered_events[{
     source_smart_contract}] // The smart contract will always trigger a
      new event
  then
   @act1 triggered_events := triggered_events ∪ {
     source_smart_contract ↦ cross_chain_event} // The smart
     contract triggers a new event related to the transaction processing
   @act2 received_transactions := received_transactions \ {
     source_smart_contract ↦ transaction} // The smart contract
     processed the transaction
 end
```

Fig. 7: Smart contract processes the transaction and triggers an event (RQ3).

```
invariants
 @inv4 gateway_pending_transactions ∈ GATEWAYS ↔
    CROSS_CHAIN_TRANSACTIONS

 event LISTEN_CC_TX_EVENT
  any cross_chain_event cross_chain_transaction
  where
   @grd1 source_smart_contract ↦ cross_chain_event ∈
     triggered_events // The smart contract has triggered an event
   @grd2 gateway ↦ source_smart_contract ∈ subscriptions //
     Exist a subscription to the smart contract events
   @grd3 gateway ↦ cross_chain_transaction ∉
     gateway_pending_transactions // The event was not already
     listened
  then
   @act1 gateway_pending_transactions :=
     gateway_pending_transactions ∪ {gateway ↦
     cross_chain_transaction} // The event is added to the cross−chain
     transactions to be processed by the gateway
   @act2 triggered_events := triggered_events \ {
     source_smart_contract ↦ cross_chain_event} // The event is
     removed from the pending events to listen
 end
```

Fig. 8: The gateway listens to the triggered event and marks it as pending (RQ4 part 1).

is received and marked as pending to be processed by the gateway. Furthermore, it is deleted from the pending events to listen. All pending transactions of the gateway are modelled as a Cartesian product between the gateways and cc-tx (see invariant). The second event, depicted in Fig. 9, represents the gateway processing cc-tx and submitting it to the target blockchain. This event can only be executed if there are pending transactions that the gateway needs to process. After the execution, the cc-tx is removed from the gateway's pending transactions, and a cc-tx is added to the target smart contract.

B. First refinement: Ethereum to Fabric interaction

This Section presents a refinement of the machine presented in Section III-A and instantiates the model with Ethereum as

```
invariants
  @inv6 received_cross_chain_transactions ∈
    CROSS_CHAIN_SMART_CONTRACTS ↔
    CROSS_CHAIN_TRANSACTIONS

event SUBMIT_CC_TX
  any cross_chain_transaction
  where
   @grd1 gateway ↦ cross_chain_transaction ∈
     gateway_pending_transactions // There is one pending cross−
     chain transaction to process
  then
   @act1 received_cross_chain_transactions :=
     received_cross_chain_transactions ∪ {target_smart_contract
     ↦ cross_chain_transaction} // The cross−chain transaction is
     received by the target smart contract
   @act2 gateway_pending_transactions :=
     gateway_pending_transactions \ {gateway ↦
     cross_chain_transaction} // Remove the cross−chain transaction
     to the pending transactions of the Gateway
end
```

Fig. 9: The gateway process the event and submits a cross-chain transaction to the target smart contract (RQ4 part 2).

the source blockchain and Hyperledger Fabric as the target blockchain. This refinement models a general-purpose one-way cross-chain smart contract interaction between Ethereum and Fabric. As Fabric requires authentication and authorisation of the incoming transactions [19], these new requirements need to be modelled on Event-B. Table II presents all the requirements that guided the refinement. These requirements correspond to the real characteristics of the selected blockchains. Asterisks (*) mark requirements that imply safety properties.

```
sets PERMISSIONS USERS

constants read write gateway_user

axioms
 @axm11 partition(PERMISSIONS, {read}, {write}) // Only two
   types of permissions exist: read and write (RQ1)
 @axm12 gateway_user ∈ USERS // The gateway has a Fabric user
```

Fig. 10: Context refinement to satisfy Fabric's requirements.

The new requirements imposed by Fabric required refining the context of the abstract machine to include new sets (i.e. permissions and users) and new constants (i.e. gateway's user and read/write permissions). Fig. 10 shows the refined context.

The machine refinement only required changes to the event that submits transactions to the target smart contract (Step 4 of Fig. 3). The refined event (Fig. 11) required two new guards: 1) restrict its execution to authenticated users, and 2) only users with write permissions can submit a cc-tx.

TABLE II: Ethereum to Fabric interaction requirements.

ID	Description
RQ1	Authorisation has two grants: READ and WRITE permissions.
RQ2	Create credentials for a user (e.g. gateway) to authenticate them.
RQ3	Grant permissions (e.g. write) to users (e.g. gateway) in Fabric.
RQ4*	Every submitted transaction to Fabric must be authenticated.
RQ5*	Every submitted transaction to Fabric must be authorised.
RQ6*	Only authorised users with write permissions can write on Fabric.
RQ7*	Only authenticated users can submit transactions to Fabric.

```
invariants
 @inv11 authenticated_users ⊆ USERS // Authenticated users are a
     subset of all the possible users
 @inv12 authenticated_transactions ∈
     received_cross_chain_transactions → authenticated_users //
     Authenticated transactions are received transactions submitted by an
     authenticated user
 @inv13 ∀ tx · tx ∈ received_cross_chain_transactions ⇒ (∃ u · u ∈
      authenticated_users ∧ tx ↦ u ∈ authenticated_transactions) //
     Every submitted transaction to Fabric must be authenticated (RQ4)
 @inv14 grants ∈ authenticated_users ↔ PERMISSIONS // Users
     with read or write permissions
 @inv15 ∀ u · u ∈ authenticated_transactions[
     received_cross_chain_transactions] ⇒ u ↦ write ∈ grants //
     Authenticated users that submitted a transaction must have write
     permissions (RQ5)

event SUBMIT_CC_TX_TO_FABRIC extends SUBMIT_CC_TX
  any user
  where
   @grd11 user ∈ authenticated_users // Only allow authenticated
     users (RQ6)
   @grd12 user ↦ write ∈ grants // Only allow authorized users (RQ7)
  then
   @act11 authenticated_transactions(target_smart_contract ↦
     cross_chain_transaction) := user // Audit user that submitted a
     transaction to a smart contract (RQ4)
end

event CREATE_GATEWAY_USER
when
 @grd1 gateway_user ∉ authenticated_users
then
 @act1 authenticated_users := authenticated_users ∪ {
     gateway_user} // Create credentials for a user (e.g. gateway) to
     authenticate them (RQ2)
end

event GRANT_PERMISSION
any permission user
where
 @grd1 permission ∈ PERMISSIONS
 @grd2 user ∈ authenticated_users
 @grd3 user ↦ permission ∉ grants
then
 @act1 grants := grants ∪ {user ↦ permission} // Users (e.g.
     gateway) can be granted permissions (e.g. write) on Fabric (RQ3)
end

event INITIALISATION extends INITIALISATION
  then
   @act11 authenticated_users := ∅
   @act12 authenticated_transactions := ∅
   @act14 grants := ∅
end
```

Fig. 11: Machine refinement to satisfy Fabric's requirements.

After event execution, the transaction is audited, and the user, the transaction, and the smart contract are recorded. Four new invariants were included to maintain the model's safety. Furthermore, a new initialisation event was specified.

C. Second refinement: Fabric to Ethereum Interaction

This second refinement models one-way cross-chain smart contract invocation from Fabric to Ethereum. As every transaction submitted to Ethereum must pay a fee to the validators [18], this behaviour must be included in the refined machine. Table III presents all the requirements that guided this refinement and correspond to real characteristics of the selected blockchains. The asterisks (*) mark the requirements that imply safety properties.

TABLE III: Fabric to Ethereum interaction requirements.

ID	**Description**
RQ1	The Gateway has an Ethereum address.
RQ2	Users can create their address on Ethereum.
RQ3	User (including the gateway) can deposit Ethers on their address.
RQ4*	The balance of each address must be equal or greater than zero.
RQ5	Submitting a transaction to Ethereum implies that the Gateway must pay a fee to the validators to process the transaction.

Accounts were modelled using a machine invariant as a partial function between Ethereum's addresses and natural numbers. Addresses are a new set defined in the refined context of the new machine. Fig. 12 shows these refined concepts.

```
axioms
  @axm11; gateway_address ∈ ADDRESS

invariants
   @inv11; accounts ∈ ADDRESS ⇸ ℕ // The balance of each address
     must be equal or greater than zero (RQ4)
```

Fig. 12: The Gateway has an address (RQ1). The balance of each address must be equal to or greater than zero (RQ4).

Creating an account (RQ1 and RQ2) and depositing Ethers in an account (RQ3) were modelled with new events of the refined machine. Fig. 13 presents their specification. The CREATE_ADDRESS_IN_ETHEREUM event enables the creation of accounts with an initial balance of zero (RQ1 and RQ2) and the DEPOSIT_CRYPTOCURRENCY_IN_ETHEREUM event allows to increment the balance of a given address with a given amount (RQ3). Furthermore, it was necessary to refine the submit transaction event (Step 4 of Fig. 3) to comply with requirement RQ5. In particular, new guards were added to validate that the gateway has enough Ethers to pay the validator's fee. Furthermore, successfully executing this event subtracts the fee amount from the gateway's account (Fig. 14).

IV. Evaluation

The specification was evaluated through three strategies: a formal verification, a functional validation using a model animation, and through a use case scenario to show its utility. Event-B specifications and animations can be found online[1].

A. Formal verification of the specification

The Event-B specification is formally verified using proof obligations (POs), which determine the correctness of Event-B models. The Rodin platform automatically generated 23 POs for all machines (see Table IV): 17 are invariant preservation PO rules (INV), and six are well-definedness PO rules (WD). INV rules verify that the defined invariants are preserved at any state change in the machine. WD rules verify that formulas

[1]https://shorturl.at/knrW4

```
event CREATE_ADDRESS_IN_ETHEREUM // Users can create their
    address on Ethereum (RQ1 and RQ2)
 any address
 where
  @grd1 address ∈ ADDRESS
  @grd2 address ∉ dom(accounts)
 then
  @act1 accounts := accounts ∪ {address ↦ 0}
end

event DEPOSIT_CRYPTOCURRENCY_IN_ETHEREUM // Users (
    including the gateway) can deposit Ethers on their address (RQ3)
 any amount address
 where
  @grd1 amount > 0
  @grd2 address ∈ dom(accounts)
 then
  @act1 accounts(address) := accounts(address) + amount
end
```

Fig. 13: Events to create an account and deposit Ethers (RQ1, RQ2 and RQ3).

```
event SUBMIT_CC_TX_TO_ETHEREUM extends
    SUBMIT_CC_TX
 any fee
 where
  @grd11 gateway_address ∈ dom(accounts)
  @grd12 accounts(gateway_address) ≥ fee // The gateway has
    enough balance to pay the validators fee (RQ5)
  @grd13 fee > 0
 then
  @act11 accounts(gateway_address) := accounts(
    gateway_address) − fee // The fee is subtracted from the gateways
    account (RQ5)
end
```

Fig. 14: Submitting a transaction to Ethereum implies that the Gateway must pay a fee to the validators to process the transaction (RQ5).

used in axioms, guards, actions, and invariants are well-defined (e.g. x divided by 0 is not a well-defined formula). All POs were automatically discharged by Rodin. Overall, it is proved that each machine's invariants, guards, and actions are defined properly and correctly by its construction.

B. Functional validation of the specification

Animation of Event-B models enables users without Event-B knowledge to validate the functional behaviour of the specification [20]. Three animations were performed using the Rodin platform with its ProB plugin and can be found in the aforementioned online materials. These animations enable the user to select the execution of events with its corresponding parameters. The safety properties check that undesirable states cannot be reached, as is shown in these animations. For example, if the gateway is not authorised, it cannot submit transactions to Fabric. These interactive animations enabled functional validation of the specification.

C. Utility validation of the specification

A use case scenario inspired by the work of Pandolfi et al. [8] was proposed to validate the utility of the specification. The use case scenario proposes a solution to prevent the gateway

TABLE IV: Proof statistics using Rodin.

Machine	Events	Invariants	POs	PO\INV	PO\WD
Abstract Machine	6	5	0	0	0
Ethereum - Fabric	8	5	12	12	0
Fabric - Ethereum	9	1	11	5	6
Total	**23**	**11**	**23**	**17**	**6**

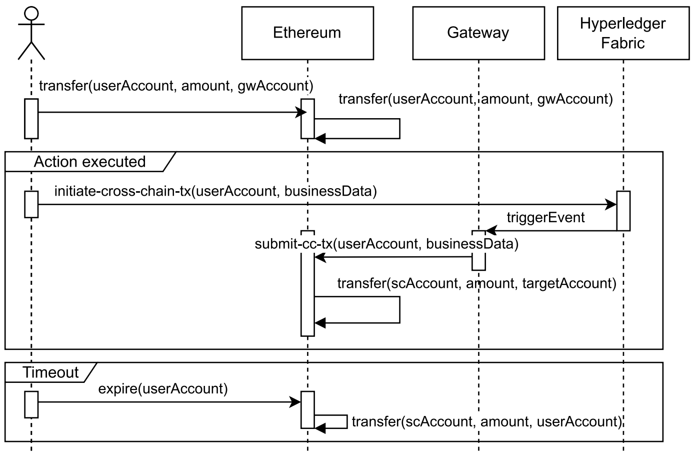
Ethereum
Gateway
Hyperledger Fabric
transfer(userAccount, amount, gwAccount)
transfer(userAccount, amount, gwAccount)
Action executed
initiate-cross-chain-tx(userAccount, businessData)
triggerEvent
submit-cc-tx(userAccount, businessData)
transfer(scAccount, amount, targetAccount)
Timeout
expire(userAccount)
transfer(scAccount, amount, userAccount)

Fig. 15: Use case scenario

from using its cryptocurrencies and leverages the payment to the user behind the cross-chain transaction. Fig. 15 presents the use-case scenario sequence diagram. Users who want to initiate a cc-tx from Fabric to Ethereum must first transfer an estimated fee to the gateway. This estimated fee is the fee that the gateway would pay if it submits the cc-tx to Ethereum.

A new Event-B machine was specified to model this use-case scenario. This new machine is a refinement of the Fabric-Ethereum machine. Table V presents the requirements that guided this refinement. Asterisks (*) mark safety properties that must be held by the system.

First, a transfer event was specified on the Fabric-Ethereum machine to allow the user to make transfers (RQ1). Initially, this event was specified in the new machine. However, an Equal proof obligation (EQL) appeared as the accounts variable was modified in a refined machine. In most cases, an EQL indicates a bad refinement [15], so it was relocated to the upper machine. This makes sense as a transaction to Ethereum can be submitted independently of the use case scenario. Fig. 16 presents the transfer specification. Second, the transfer event was refined in the new machine to save the estimated fee. This enabled adding a guard to the SUBMIT_TRANSFER_TRANSACTION_IN_ETHEREUM event to check that the gateway has the amount transferred by the user in its account (RQ2). Fig. 17 presents these refined events.

TABLE V: Use case scenario requirements.

ID	Description
RQ1	The user can transfer an estimated fee to the gateway account.
RQ2*	The gateway only submits the cc-tx to Ethereum after the user transfers the estimated fee to its account. The estimated fee must be equal to or greater than the fee of Ethereum's validators
RQ3*	The gateway does not own any cryptocurrencies in his account to pay for cc-tx.

```
event SUBMIT_TRANSFER_TRANSACTION_IN_ETHEREUM //
    The user can transfer an estimated fee to the gateways account (RQ1)
any validator_fee user_address transfer_amount
  where
   @grd1 user_address ∈ dom(accounts) // The user has an account
   @grd2 accounts(user_address) ≥ validator_fee +
     transfer_amount // The user has enough balance to do the transfer
     and pay the validator's fee
   @grd3 validator_fee > 0
   @grd4 transfer_amount > 0
   @grd5 gateway_address ∈ dom(accounts) // The gateway has an
     account
   @grd6 ({user_address,gateway_address} ⩤ accounts)∪{
     gateway_address ↦ accounts(gateway_address)+
     transfer_amount}∪{user_address ↦ accounts(user_address)
     − transfer_amount − validator_fee}∈ADDRESS ⇸ ℕ
  then
   @act1 accounts := ({user_address, gateway_address} ⩤
     accounts) ∪
      {gateway_address ↦ accounts(gateway_address) +
     transfer_amount} ∪
      {user_address ↦ accounts(user_address) −
     transfer_amount − validator_fee} // Subtracts the transfer amount
     and fee from the user's account and add the transfer amount to the
     gateways account
 end
```

Fig. 16: Transfer event specification (RQ1).

```
invariants
  @inv31 estimated_cross_chain_cost ∈ ℕ


event SUBMIT_TRANSFER_TRANSACTION_IN_ETHEREUM
    extends
    SUBMIT_TRANSFER_TRANSACTION_IN_ETHEREUM
  then
   @act31 estimated_cross_chain_cost := transfer_amount
end
event SUBMIT_CC_TX_TO_ETHEREUM extends
    SUBMIT_CC_TX_TO_ETHEREUM
  where
   @grd31 fee ≤ estimated_cross_chain_cost // Check that the
     estimated fee is greater or equal to the validators fee (RQ2)
end
```

Fig. 17: Gateway has an estimated fee in its account (RQ2).

Finally, to ensure that the gateway cannot pay the cc-tx with its cryptocurrencies (RQ4), a new guard was added to the deposit event to prevent deposits from the gateway (see Fig. 18).

Rodin generated two new INV POs for the new transfer event. All POs were automatically discharged by Rodin.

Overall, the existing model could be extended to include new events, a use case scenario could be defined (through a new refinement), and both models were defined correctly by construction. Furthermore, safety properties were validated by event guards, preventing from reaching an undesirable state.

D. Discussion and limitations

One of the main findings of this study was that Event-B was a suitable method to formally specify a gateway-based interoperability solution. First, it was possible to specify an abstract model independent of the source and target blockchains that shows its behaviour for one-way cross-chain smart contract invocation. Second, it was possible to refine this abstract

```
event DEPOSIT_CRYPTOCURRENCY_IN_ETHEREUM extends
    DEPOSIT_CRYPTOCURRENCY_IN_ETHEREUM
  where
   @grd31 address ≠ gateway_address // The gateway cannot
     deposit in its own account (RQ3)
end
```

Fig. 18: Prevent deposits from the gateway (RQ3).

model and specify two new models to include the behaviour of two specific blockchains: Ethereum and Fabric. Third, the specification could be evaluted using three strategies: formal verification, functional validation and functional utility. Formal verification enabled us to verify the model correctness and identified safety properties that prevent undesirable state changes (e.g. allowing unauthorised cross-chain transactions to be submitted to Fabric). The validation through animation modelling enabled us to validate the functional behaviour of the specification and provide the means to understand the gateways' behaviour without Event-B knowledge. Finally, the use case scenario showed the utility of the specification, showing how a concrete specification can be extended to specify a use case scenario and prove its safety properties.

Overall, the results of the study provided preliminary results to answer the proposed research question: **How can the gateway-based approach be formalised with Event-B?** This study can claim that it was possible to model a specific gateway-based interoperability solution with Event-B formally. The Event-B method was adequate in modelling it with two selected blockchains: Ethereum and Fabric. The model obtained was straightforward, with little intricacies or hard demands that pushed the limits of the method. Furthermore, it proved to be extensible for specific use case scenarios and provided a communication tool (i.e. animations) with domain experts without Event-B knowledge. Finally, the results showed that the gateway behaviour is heavily dependent on the blockchains involved, as little behaviour could be abstracted (see Table I). This could possibly be due to the nature of the blockchains involved. Ethereum is a permissionless blockchain, and Fabric is permissioned. Further work with other blockchains is needed to confirm this.

On the other hand, there exist some limitations related to the scope of the work and limitations of the method. Event-B is a suitable method to verify the models' functional correctness and safety properties. In turn, verifying liveness properties is not a straightforward task, and there are limitations on how to describe them. In particular, to preserve liveness properties in refined machines as we use in this work [10], [22]. Furthermore, this work did not consider transaction finality, and it was left as future work. Although it was possible to specify a specific interoperability solution that followed the gateway approach, we cannot conclude that all interoperability gateways can be modelled with this method. Further work needs to be performed to generalise that this method suits the gateway-based approach. For example, with other gateway solutions (e.g. Hermes [5], Weaver [23]). Second, this work only modelled two blockchains (i.e. Ethereum and Fabric). Additional work needs to be performed with other blockchains to validate its suitability. Furthermore, the model was performed by this work's main author and validated by the other authors. There was no external validation of the model. In addition, the model included one-way interactions, leaving request-reply interactions out of scope. Finally, real use case scenarios need to be modelled to ensure that the method is adequate.

V. Related work

Efforts to formalise blockchain interoperability solutions are scarce. To the best of our knowledge, our work is the first approach to formalise them with Event-B. Other authors use other formal methods or Event-B to formalise smart contract development, but none use Event-B to formalise an interoperability solution. Therefore, we discuss the existing work and highlight the approaches, similarities, and results.

Wei et al. [12] is the work most related. The authors formalised the main components of the IBC protocol with TLA+ and verified the model with the requirements that the IBC protocol must meet from the official documentation. They focused on security issues that may occur during handshake and packet transmission by on-chain and off-chain components. In fact, some issues related to incomplete handshakes, timeout conditions, and abnormal channel states were reported and notified to the community. Furthermore, they made recommendations to solve these problems. This work is similar to ours, as it formalises interoperability solutions to maintain functional correctness and safety properties. In particular, the middleware must comply with the properties to transfer messages from the source blockchain to the target blockchain. Our work complements this work and uses another model (i.e. Event-B) to specify interoperability solutions besides TLA+. However, one drawback of our approach is that Event-B cannot express liveness properties. TLA+ can be composed with the TLC model checker to prove liveness properties.

Babu et al. [14] formalises the Burn-to-Claim protocol, an asset transfer protocol between two blockchains. The authors developed the model using the CSP# language and verified it with the PAT model checker. Using PAT, they examined the protocol's correctness, security, and atomicity properties. This work also shares similarities with ours since it uses a formal method to verify certain properties of an interoperability protocol. However, the authors focused on the protocol steps required in each blockchain and did not include the required middleware (i.e. gateway) that relays messages between blockchains as we do. A new refinement to our machines may be performed to model the Burn-to-Claim protocol in Event-B and consider the middleware. This improvement may enrich the formalisation of the protocol.

Nehaï et al. [13] proposed a formalisation using TLA+ for cross-chain swaps. With this model, they prove the safety and liveness properties of the protocol with Byzantine participants. However, this work also focused on the actions performed on each blockchain and did not consider the middleware to relay messages. Our approach considered this middleware (i.e. gateway) but left out of scope liveness and security properties and focused only on the functional correctness of the model.

On the other hand, several works used Event-B to verify correctness and safety properties when writing smart contracts in Solidity. Singh et al. [16] proposed EB2Sol, a tool to generate smart contracts written in solidity from an Event-B specification, to maintain functional correctness and safety properties by construction. Zhu et al. [24] developed a tool that transfers a subset of the solidity language to an Event-B specification. They applied the tool in a public Honeypot smart contract and proved its functional correctness and safety properties. Finally, Lahbib et al. [17] used Event-B as a formal method to verify smart contracts written in solidity. In particular, they manually modelled a smart contract to verify their safety, correctness, functional accuracy, and compliance with the requirements specification. Although these works do not directly relate to our work, they may complement it in the future. Further refinements of our model that specify low-level details of the behaviour of the source and target smart contracts may be automatically translated to Solidity.

Overall, our work complements the existing body of knowledge and shows that it is possible to formalise certain properties (e.g. safety and correctness by construction) of a specific gateway-based solution with Event-B. Further work and comparisons are needed between Event-B, TLA+, PAT and other formal methods to evaluate the most suitable method for this type of system. To the best of our knowledge, our work was the first approach to formalise a gateway-based solution with Event-B, going a step further in the research process.

VI. Conclusions and future work

One of the key findings of this study was that it was possible to model a specific gateway-based interoperability solution with Event-B. This finding enable Event-B to be included to the existing candidate list of formal methods that can be used to formalise interoperability solutions (i.e. TLA+ and PAT). Second, the method resulted effective for this solution, providing a straightforward specification without pushing the model limits. Third, the proposed layered modelling through refinement and the three evaluation strategies may serve as guidelines for future studies that use Event-B to formalise interoperabiliy solutions. Fourth, formalising the gateway showed that its behaviour is heavily dependent on the blockchains involved, and little behaviour could be abstracted as a common behaviour that is not dependent on the blockchains. Overall, this study presented promising preliminary results using Event-B to specify gateway-based solutions formally.

Future work includes verifying liveness properties, considering transaction finality, testing the specification with other blockchains, specify other interactions besides one-way cross-chain smart contract invocations, and applying the method to real use case scenarios.

References

[1] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, “A survey on blockchain interoperability: Past, present, and future trends,” *ACM*

Comput. Surv., vol. 54, no. 8, pp. 1–41, Oct. 2021. [Online]. Available: https://doi.org/10.1145/3471140

[2] V. A. Siris, P. Nikander, S. Voulgaris, N. Fotiou, D. Lagutin, and G. C. Polyzos, "Interledger approaches," *IEEE Access*, vol. 7, pp. 89 948–89 966, Jul. 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2926880

[3] G. Llambías, L. González, and R. Ruggia, "Blockchain interoperability: a feature-based classification framework and challenges ahead," *CLEI Electron. J.*, vol. 25, no. 3, pp. 1–29, 2022. [Online]. Available: https://doi.org/10.19153/cleiej.25.3.4

[4] T. Hardjono, A. Lipton, and A. Pentland, "Toward an interoperability architecture for blockchain autonomous systems," *IEEE Transactions on Engineering Management*, vol. 67, no. 4, pp. 1298–1309, Nov. 2020. [Online]. Available: https://doi.org/10.1109/TEM.2019.2920154

[5] R. Belchior, A. Vasconcelos, M. Correia, and T. Hardjono, "Hermes: Fault-tolerant middleware for blockchain interoperability," *Future Generation Computer Systems*, vol. 129, pp. 236–251, Apr. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X21004337

[6] E. Abebe, D. Behl, C. Govindarajan, Y. Hu, D. Karunamoorthy, P. Novotny, V. Pandit, V. Ramakrishna, and C. Vecchiola, "Enabling enterprise blockchain interoperability with trusted data transfer (industry track)," in *Proceedings of the 20th International Middleware Conference Industrial Track*, Davis, CA, USA, Dec. 2019, p. 29–35. [Online]. Available: https://doi.org/10.1145/3366626.3368129

[7] G. Llambias *et al.*, "Gateway-based Interoperability for DLT," *CLEI Electron. J.*, vol. 26, no. 2, pp. 1–24, Feb. 2023. [Online]. Available: https://doi.org/10.19153/cleiej.26.2.5

[8] S. Pandolfi, E. González, M. Castro, G. Llambías, L. González, and R. Ruggia, "Interoperability between dlt following a gateway-based approach: The case of ethereum and hyperledger fabric," in *2023 XLIX Latin American Computer Conference (CLEI)*, La Paz, Bolivia, Oct. 2023, pp. 1–10. [Online]. Available: 10.1109/CLEI60451.2023.10346168

[9] B. Pillai *et al.*, "Cross-blockchain technology: Integration framework and security assumptions," *IEEE Access*, vol. 10, pp. 41 239–41 259, Apr. 2022, doi: 10.1109/ACCESS.2022.3167172.

[10] C. Zhu, M. Butler, C. Cirstea, and T. S. Hoang, "A fairness-based refinement strategy to transform liveness properties in event-b models," *Science of Computer Programming*, vol. 225, no. C, pp. 1–23, Jan. 2023. [Online]. Available: https://doi.org/10.1016/j.scico.2022.102907

[11] R. Belchior, L. Riley, T. Hardjono, A. Vasconcelos, and M. Correia, "Do you need a distributed ledger technology interoperability solution?" *Distrib. Ledger Technol.*, vol. 2, no. 1, pp. 1–37, Mar. 2023. [Online]. Available: https://doi.org/10.1145/3564532

[12] Q. Wei, X. Zhao, X.-Y. Zhu, and W. Zhang, "Formal analysis of ibc protocol," in *2023 IEEE 31st International Conference on Network Protocols (ICNP)*, Reykjavik, Iceland, Oct. 2023, pp. 1–11. [Online]. Available: https://doi.org/10.1109/ICNP59255.2023.10355573

[13] Z. Nehaï, F. Bobot, S. Tucci-Piergiovanni, C. Delporte-Gallet, and H. Fauconnier, "A tla+ formal proof of a cross-chain swap," in *Proceedings of the 23rd International Conference on Distributed Computing and Networking*, Delhi, AA, India, 2022, p. 148–159. [Online]. Available: https://doi.org/10.1145/3491003.3491006

[14] B. Pillai, Z. Hóu, K. Biswas, and V. Muthukkumarasamy, "Formal verification of the burn-to-claim blockchain interoperable protocol," in *Formal Methods and Software Engineering*, Brisbane, QLD, Australia, Nov. 2023, pp. 249–254. [Online]. Available: https://doi.org/10.1007/978-981-99-7584-6_15

[15] K. Robinson, *System modelling & design using Event-B*, 2012.

[16] N. K. Singh, A. M. Fajge, R. Halder, and M. I. Alam, "Formal verification and code generation for solidity smart contracts," in *Distributed Computing to Blockchain*. Cambridge, MA, USA: Academic Press, 2023, pp. 125–144. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780323961462000280

[17] A. Lahbib, A. Ait Wakrime, A. Laouiti, K. Toumi, and S. Martin, "An event-b based approach for formal modelling and verification of smart contracts," in *Advanced Information Networking and Applications*, Caserta, Italy, Apr. 2020, pp. 1303–1318. [Online]. Available: https://doi.org/10.1007/978-3-030-44041-1_111

[18] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform," Jun. 2014, (accessed Jan. 2024). [Online]. Available: https://ethereum.org/en/whitepaper/

[19] 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 permissioned blockchains," in *Proc. 13th EuroSys Conf.*, Porto, Portugal, April 2018, pp. 1–15. [Online]. Available: https://doi.org/10.1145/3190508.3190538

[20] J.-R. Abrial, *Modeling in Event-B: system and software engineering*. Cambridge, United Kingdom: Cambridge University Press, 2010.

[21] J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin, "Rodin: an open toolset for modelling and reasoning in Event-B," *Software Tools for Technology Transfer*, vol. 12, no. 6, pp. 447–466, 2010. [Online]. Available: https://doi.org/10.1007/s10009-010-0145-y

[22] P. Rivière, N. K. Singh, Y. Aït-Ameur, and G. Dupont, "Formalising liveness properties in event-b with the reflexive eb4eb framework," in *NASA Formal Methods 2023*, Houston, TX, USA, May. 2023, pp. 312–331. [Online]. Available: https://doi.org/10.1007/978-3-031-33170-1_19

[23] Weaver Team, "Weaver framework," (accessed Jan. 2024). [Online]. Available: https://labs.hyperledger.org/weaver-dlt-interoperability/docs/external/introduction

[24] J. Zhu, K. Hu, M. Filali, J.-P. Bodeveix, J.-P. Talpin, and H. Cao, "Formal simulation and verification of solidity contracts in event-b," in *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, Madrid, Spain, Jul. 2021, pp. 1309–1314. [Online]. Available: https://doi.org/10.1109/COMPSAC51774.2021.00183