Specifying Blockchain-Based Resource-Exchange Systems by Business-Level Users Using a Generic Easy-To-Use Framework

Kushal Soni[1] and Olga De Troyer[2]

[1] Computer Science Department, Vrije Universiteit Brussel, Brussels, Belgium
`Kushal.Soni@vub.be`
[2] Computer Science Department, Vrije Universiteit Brussel, Brussels, Belgium
`Olga.DeTroyer@vub.be`

Abstract. Blockchain technology has been rapidly emerging in the past years. The notion of decentralization, enabling ecosystems that provide true ownership of resources without requiring a trusted third party, is gaining interest and is applied in use cases daily. Such use cases vary from simple currency-exchange applications to complex smart contract applications. Designing, building, and deploying blockchain-based applications mostly requires software developers. This introduces a technological burden for organizations who would like to benefit from such systems in their own use case(s) but lack time, financial means, or qualified people. To overcome this burden, we developed a software framework allowing an easy and quick setup of blockchain-based systems for use cases dealing with the exchange of resources. To set up such a system, an easy-to-use user interface is provided that allows business-level people to give the specifications of the system without the need for technical software-, blockchain- or smart contract knowledge. From these specifications an implementation is generated. In this paper, we present the architecture of the framework and discuss the principles used, and the user interface developed for specifying such use cases.

Keywords: Blockchain, Smart Contracts, Generic Framework, Specification System, Easy-To-Use, Resource Exchange

1 Introduction

More and more businesses and organizations are offering digital services, such as online shopping, cloud storage, streaming services to increase user convenience and revenue. However, this implies the need for IT infrastructures to store and maintain related user data and/or resources. Currently, users of these services must rely on the goodwill of the businesses and organizations, and assume that these will reliably manage their data and resources. This implies that users are lacking true ownership of their data and resources. Moreover, since providing such services requires (significant) data storage, (intense) computing power and IT experience, many businesses and organizations prefer to rely on specialized cloud-based infrastructure providers, such as Amazon, Google, DigitalOcean. By leveraging such services, they save both time and financial investments, which would otherwise be required for hardware and expensive software developers. However, consequently, they give away control of their data. As a result, users of the services are also forced to expose their personal data, as well as hand over full control of it, not only to the businesses or organizations providing the services, but also to the underlying cloud providers, managing the services. This leaves users in a vulnerable “forced-to-trust” position. The use of blockchain technology, embedded with smart contracts, would be a good solution to deal with these issues. In [1], blockchain technology is proposed to introduce verifiable true ownership in ecosystems, whilst ensuring reliability, robustness, integrity of data, and prevention of data loss. However, to develop blockchain-based applications, one needs to rely on software developers. This introduces a technological burden for organizations who would like to benefit from such systems in their own use case(s) but lack time, financial means, or qualified people. To overcome this barrier, we propose a software framework, usable by business-level people, which allows an easy and quick setup of blockchain-based systems with which organizations and their users can exchange resources.

The proposed framework has the advantage that it allows business-level people without technical software-, blockchain- or smart-contract knowledge, to set up custom use cases by simply providing the required information. The actual blockchain-based use-case system is generated based on the given information. In addition, the framework also generates a web application for the use case, supporting the actual exchange of resources between the involved organizations and users. The use of this framework saves time and resources, and allows all organizations involved in a use case (independent of their size) to participate.

Creating the proposed framework posed several challenges. The first challenge was to provide a generic approach that can deal with different types of use cases. Since use cases for the exchange of resources can vary strongly (in terms of the types of resources and rules involved), software needs to be constructed that allows to enter and capture all possibilities and which can generate smart contracts adapted to the specific specifications of the individual use cases. A second challenge pertained to the programming constructs available for the development of blockchain applications. These are still in an early stage: standard object-oriented programming constructs are not available and debugging blockchain applications is cumbersome (the well-known "revert" or "out of gas" errors are not very descriptive).

In this paper, the focus is on the component that allows a businessperson to specify a use case. However, to provide the context, we also present the overall architecture of the framework. Section 2 introduces background and in section 3, we discuss related work. The overall framework is presented in section 4. In section 5, we report on the user study performed as formative evaluation. The paper ends with conclusions and future work.

2 Background

A blockchain is a distributed, decentralized kind of database, allowing data storage not controlled by a single party [2, 3]. The data stored in blockchains is distributed across the nodes of a peer-to-peer network. Based on the purpose, transaction costs, and the popularity, participants select a network to join. Electronic cash systems and smart contracts are two examples of the most common use cases building on top of a blockchain. For instance, the most popular currency for digital payments is bitcoin (BTC) [4], which runs on the bitcoin network. Ethereum is the most popular protocol used for establishing and interacting with smart contracts.

Smart contracts are programs that reside on the blockchain, allowing traceable and immutable data modifications, such as transfer of ownership of resources based on defined application logic [5, 6]. Ethereum is the most popular network to run smart contracts, using ether (ETH) as native currency. Binance (Smart) Chain [7], Cardano [8], Polkadot [9] and Solana [10] are other examples.

Solidity[1] is the programming language used to write smart contracts for Ethereum. Although it has similar characteristics as other programming languages, it has quite some limitations. Calling a function in Solidity corresponds to executing a transaction on the Ethereum blockchain. Hence, the more instructions within a single function, the more transaction fees (gas) one will have to pay [11]. Therefore, when using Solidity, one needs to take code efficiency and contract size into account. Also, not all common programming constructs are available in Solidity. Truffle [12] and Embark [13] are developer frameworks which ease the testing and development of smart contracts. They are intended to support software developers.

Tokens are digital assets on top of a cryptocurrency or blockchain [14], often used to represent ownership of a resource. The ERC Token standard [15] has a large number of token types. The most popular ones can be summarized into two base categories: Fungible Tokens and Non-Fungible Tokens. The ERC-20 Standard [16] is the most widely used and general token standard for Fungible Tokens. In practice, this token is mostly used as a form of payment in smart contracts. The ERC-721 Standard [17] concerns Non-Fungible tokens, where each token is distinct (aka non-fungible). In practice, such tokens are used to represent ownership of unique assets (e.g., artworks or academic degrees). The ERC-1155 Standard [18] allows for the management of any combination of Fungible and Non-Fungible Tokens.

3 Related Work

In this section, we explore related work. To the best of our knowledge, no frameworks exist that ease the setup of systems with which end users are able to exchange resources across organizations, through the use of an intuitive, easy to use interface that allows the specification of permissions and rules without the necessity for knowledge about blockchain or smart contracts. However, there are tools, such as generators and marketplaces available, allowing to quickly create tokens on the blockchain without the need for any blockchain or smart contract knowledge, as we will discuss below (section

[1] https://docs.soliditylang.org/en/v0.8.6/

3.1). Additionally, we discuss specification tools designed for more complex systems (section 3.2).

3.1 Generators and Marketplaces for Non-Programmers

Minacori has created a tool for specifying and deploying Fungible ERC-20 Tokens on Ethereum [19]. Its purpose is to provide people with the ability of tokenizing their ideas without coding or paying large amounts for it. One can specify the initial and total supply of tokens, as well as whether the total supply can be increased or decreased at a later stage. The tool requires users to have the browser extension MetaMask [20] installed, which is a web3.js-based [21] electronic wallet that allows purchasing, selling, sending and receiving of tokens, as well as engagement in other types of transactions. Minacori also created tools for the creation of BEP-20 tokens, the Binance Smart Chain version of ERC-20 tokens on Ethereum [22].

Opensea is a marketplace for Non-Fungible Tokens [23]. It allows to create, buy, sell, and auction tokens. After creating a collection, one can create items provided with images, videos or 3D models, individual properties and more.

3.2 Smart Contract Specification for Business-Level Users

Astigarraga et al. [24] propose a framework for specifying smart contracts with a controlled English business-level rules language (called BCRL). The authors state that business-level users should be able to understand, create, and modify smart contracts or at least large portions of them. BCRL seems to be quite powerful, but this has the disadvantage that the language is also quite complex and will require training to be able to use it. Defining rules is possible with their smart editor, but it is not clear how much support this editor provides for non-trained persons. Compared to our work, this work seems to be more focused on complex business cases. In addition, code generation is only for the Hyperledger Blockchain fabric, which limits the scope to permissioned blockchains.

In [25] Caterpillar is described, a blockchain-based BPMN[2] execution engine. The Ethereum blockchain is used and smart contracts are generated by a BPMN-to-Solidity compiler. Similar to the previous work, this work focusses on complex business applications.

In [26], a framework for the auto-generation of smart contracts is presented. It uses ontologies and semantic rules to represent the domain knowledge of the use case. A smart contract template serves as the basis for the final smart contract. While our framework is tailored towards resource exchange, this ontology approach can be used for a much broader range of applications. However, this power comes with the price that first an ontology needs to be developed and the sematic rules need to be expressed in SWRL, two tasks that require time and technical expertise.

SPESC [27] is a specification language for smart contracts aiming to ease the process of smart contract creation in a collaborative environment. To do so, the authors introduce domain concepts such as parties, terms, conditions, and transactions and involve them in a custom specification language to set up smart contracts. However,

[2] Business Process Model and Notation – https://www.bpmn.org/

though the used language does not seem to be very complex, it is a mix of class-type (java, python, …) and query-type (sql) languages. Therefore, compared to our work, the specification still requires basic programming skills.

UDL-SC [28] is a description language for smart contracts that facilitates the analysis of smart contracts and its blockchain. It also helps programmers to better understand smart contracts that are not open source (since smart contract code is compiled to bytecode, which is hard to interpret). Like our framework, UDLSC makes it easier to understand (deployed) smart contracts, but our framework allows to generate the smart contracts, as opposed to theirs.

4 Framework

The proposed framework is intended for the development of blockchain-based systems for supporting the exchange of resources across different organizations and end users. The framework allows business-level people, without technical IT knowledge to quickly set up such a system. In this section, we explain the architecture of the framework (see Fig. 1), the different modules and their interaction, and the communication of the framework with the blockchain network. We end this section with justifying our design decisions by discussing their advantages.

Note that we use two different terms to distinguish between the people interacting with the framework to set up a use case system (i.e., business-level people) and the people interacting with a generated use case system. We call the latter ones “end users” and the first ones “users”.

4.1 Overall Architecture

The framework consists of two independent layers. As shown in Fig. 1, Layer 1 contains two components. The “Use Case Specification” component (1) is a web app that allows users to create use cases by giving their specifications. These specifications are passed to Layer 2 (2), which generates the necessary smart contracts (3). These smart contracts will be deployed on a blockchain network ((4) and (5)) chosen during setup of the use case, e.g., the Ethereum Network. The “Use Case Interaction” component is a web app generated for each specified use case from the specifications given (6) and allowing end users to consult their digital resources and transfer these resources. It calls the generated smart contracts functions, which result in transactions (8) and state changes (such as balance of currencies or change of ownership of resources) (9) on the blockchain.

4.2 Layer 1 – Use Case Specification

The specification component provides business-level people a user interface to formulate the specifications of their use case. The component is implemented as a responsive web application, storing use case specifications by running the Django[3]

[3] https://www.djangoproject.com/

framework as backend and using React[4] as frontend. The tool allows the user to go (back and forth) through different steps, each gathering different types of information.

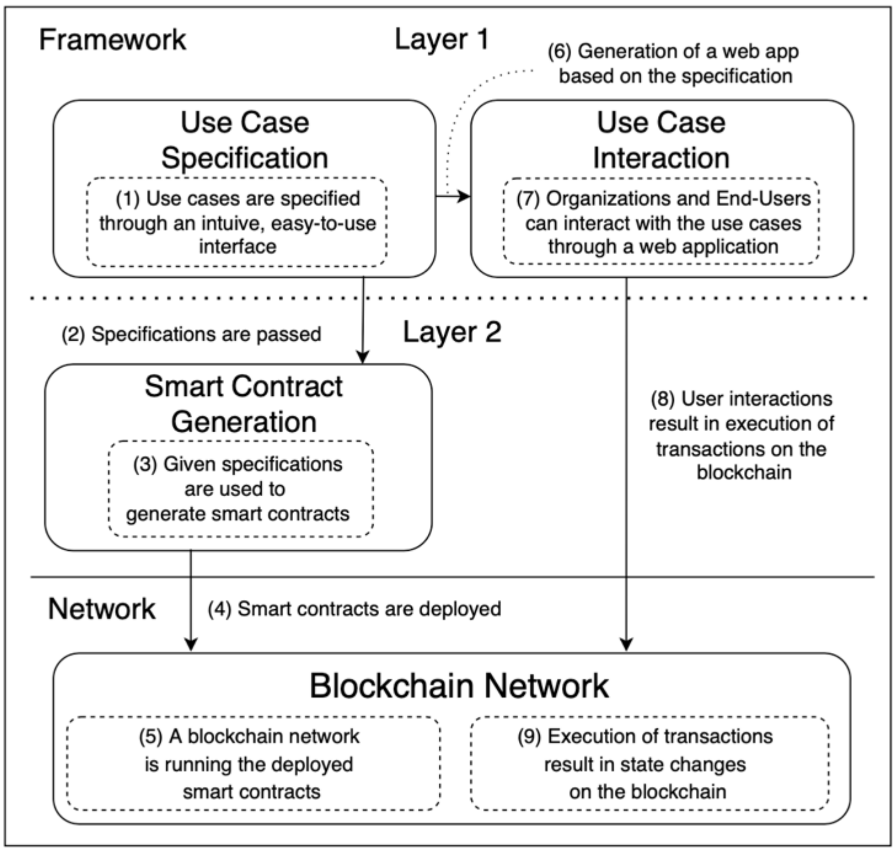

Framework
Layer 1
(6) Generation of a web app based on the specification
Use Case Specification
(1) Use cases are specified through an intuive, easy-to-use interface
Use Case Interaction
(7) Organizations and End-Users can interact with the use cases through a web application
(2) Specifications are passed
Layer 2
Smart Contract Generation
(3) Given specifications are used to generate smart contracts
(8) User interactions result in execution of transactions on the blockchain
Network
(4) Smart contracts are deployed
Blockchain Network
(5) A blockchain network is running the deployed smart contracts
(9) Execution of transactions result in state changes on the blockchain

Fig. 1. Architecture, network, and interactions

Now, we present the concepts used for specifying a use case specification and how they are specified in the user interface. The UML class diagram is given in Fig. 2. Examples of the user interface are given in Fig. 3 to Fig. 6. Apart from naming the use case (step 1), the following needs to be specified:

Organizations. All organizations participating in the resource exchange of the use case should be specified by providing a name and a unique identifier (step 2). The VAT-nr, a description, email address, website, and postal address are optional.

[4] https://reactjs.org/

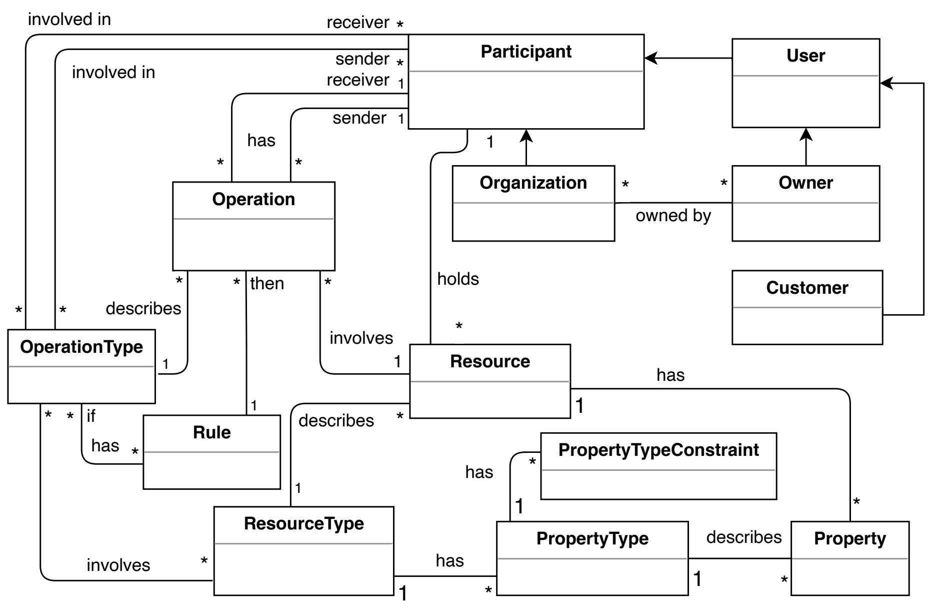
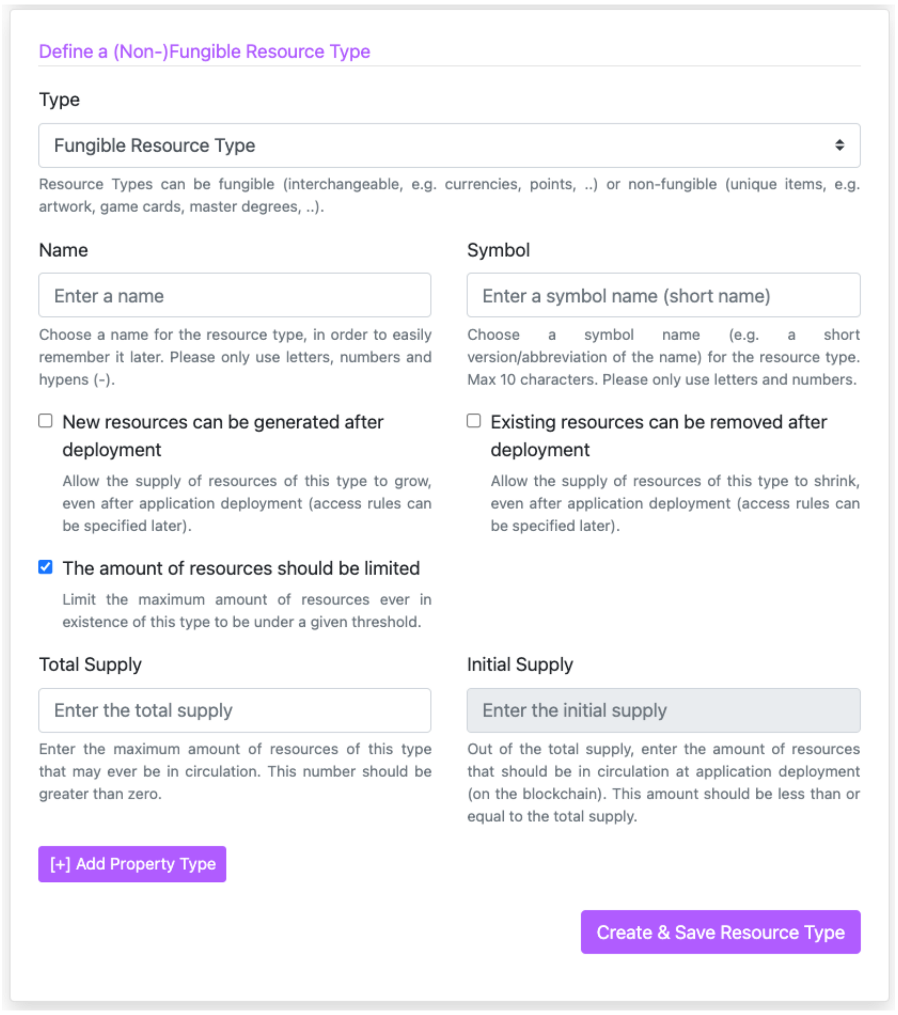


Fig. 2. UML Class diagram for the specification component

Owners. Organizations need to be represented by one or more individuals who can sign transactions and messages in the name of the organization; they are called owners and should be specified by a unique name (step 3); the option "Select All" easily allows to appoint a single owner to all organizations involved in the use case.

End Users. If only particular people can participate in the resource exchange of the use case, they should be specified by means of a unique name (step 4). In case that anyone is allowed to participate, this step can be skipped.

Resource Types. Each resource is either a FRT (Fungible Resource Type) or a NFRT (Non-Fungible Resource Type). FRTs can be limited in supply. If so, the total supply should be given. Otherwise, one must give the initial supply. In the latter case, the supply can be increased at a later stage, after contract deployment, and the initial supply will be distributed among the participating organizations according to the given ratios. One should specify whether resources of a specific type can be removed after contract deployment by (an) appointed individual(s). NFRTs are similar to NFTs mentioned in section 2. For example, the resource type "Academic Degrees" (a NFRT) could be used for the resource "A master's degree, awarded to John Doe by MIT". Resource types should have a unique name and symbol. All needed resource types should be defined (step 5 - see Fig. 3).


Define a (Non-)Fungible Resource Type

Type

Fungible Resource Type

Resource Types can be fungible (interchangeable, e.g. currencies, points, ..) or non-fungible (unique items, e.g. artwork, game cards, master degrees, ..).

Name

Enter a name

Choose a name for the resource type, in order to easily remember it later. Please only use letters, numbers and hypens (-).

Symbol

Enter a symbol name (short name)

Choose a symbol name (e.g. a short version/abbreviation of the name) for the resource type. Max 10 characters. Please only use letters and numbers.

☐ New resources can be generated after deployment

Allow the supply of resources of this type to grow, even after application deployment (access rules can be specified later).

☐ Existing resources can be removed after deployment

Allow the supply of resources of this type to shrink, even after application deployment (access rules can be specified later).

☑ The amount of resources should be limited

Limit the maximum amount of resources ever in existence of this type to be under a given threshold.

Total Supply

Enter the total supply

Enter the maximum amount of resources of this type that may ever be in circulation. This number should be greater than zero.

Initial Supply

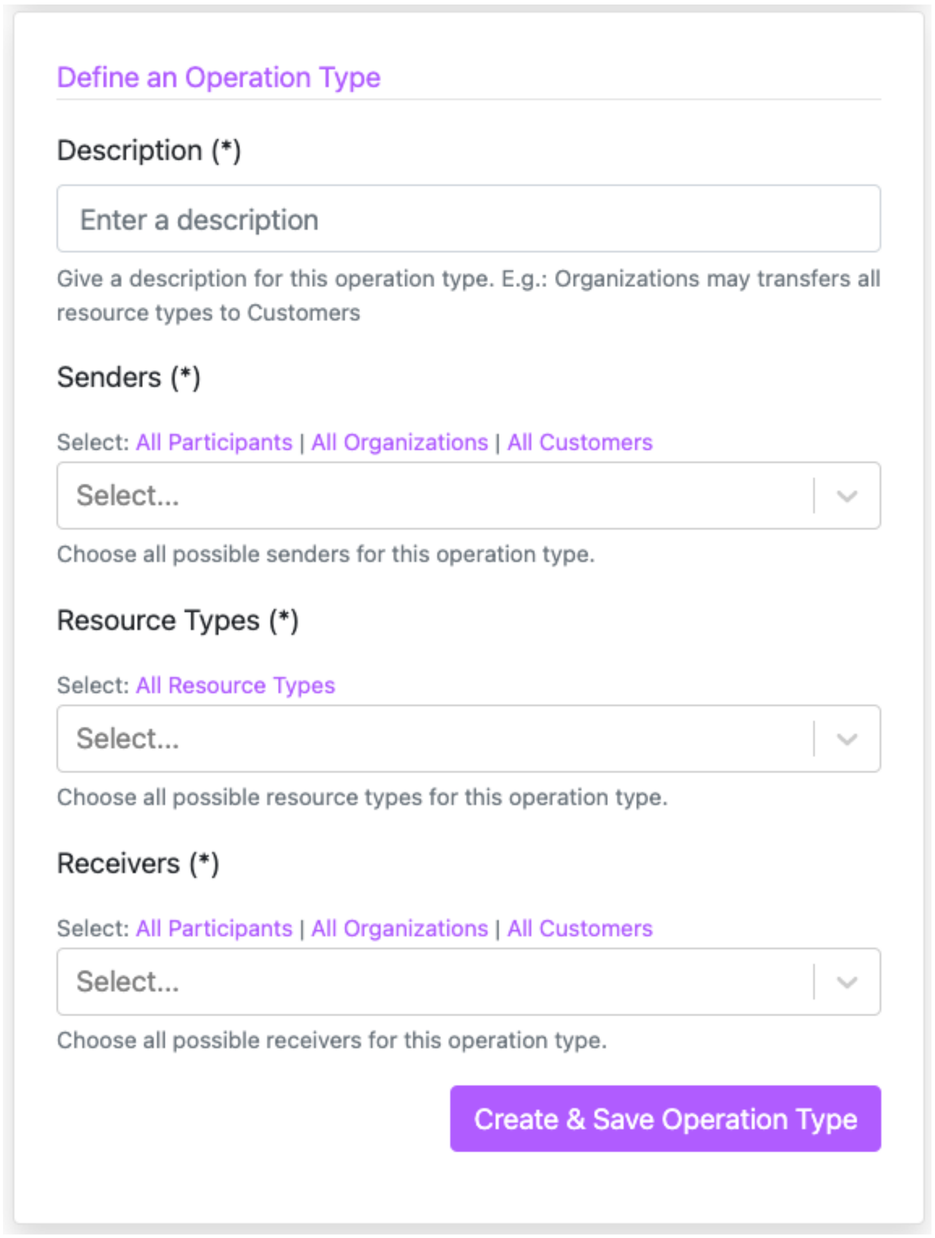
Enter the initial supply

Out of the total supply, enter the amount of resources that should be in circulation at application deployment (on the blockchain). This amount should be less than or equal to the total supply.

Fig. 3. User Interface: Step 5 - Define Resource Types

Property Types. In addition to the required properties, resources may have some optional properties. These properties are described by property types. For example, we can add the overall score and degree of distinction as additional properties to the resource type "Academic Degree", allowing to have the resource "A master's degree, awarded to John Doe by MIT", with 85/100 as overall score and great distinction as degree of distinction". Property types are defined in step 5 by means of a unique name, a data type, and an optional constraint pattern to limit its possible values.

Operation Types. An operation will correspond to an immutable data modification in the network, such as the change of ownership of a resource. An operation type defines the properties of similar operations, such as the possible sender(s) and receiver(s), and resource types. All types of operations that may occur in the use case should be defined in step 6 (Fig. 4). Note that only transactions compliant with a defined operation type can be executed after deployment.


Define an Operation Type

Description (*)

Enter a description

Give a description for this operation type. E.g.: Organizations may transfers all resource types to Customers

Senders (*)

Select: All Participants | All Organizations | All Customers

Select...

Choose all possible senders for this operation type.

Resource Types (*)

Select: All Resource Types

Select...

Choose all possible resource types for this operation type.

Receivers (*)

Select: All Participants | All Organizations | All Customers

Select...

Choose all possible receivers for this operation type.

Fig. 4. User Interface: Step 6 - Define Operation Types

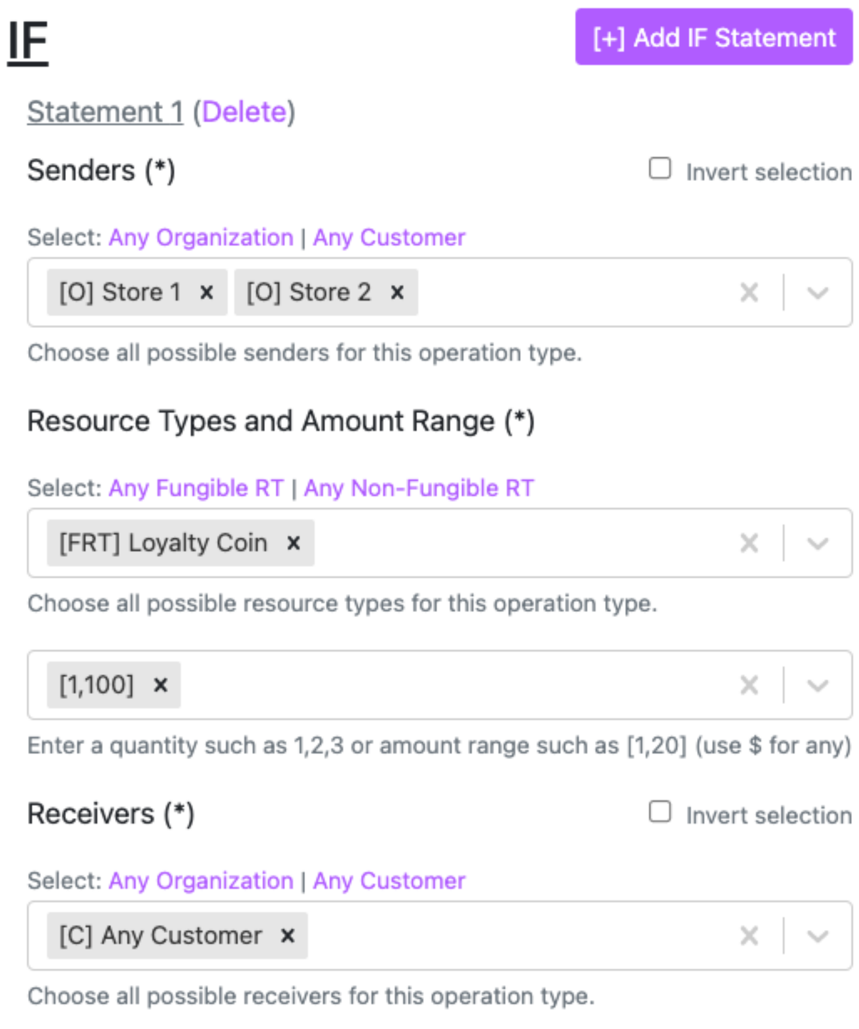
Rules. A rule allows to define requirements and constraints for operation types and operations, i.e., what type of operations should take place in order to execute (an)other specified operation(s). A rule contains one or more IF- and THEN statements. An IF statement has the purpose of looking for specific executed operations on the network. For this, it uses a set of conditions involving constraints on the sender(s), resource type(s) and amount(s), and the receiver(s) of an operation. Multiple senders, receivers, resource types and an amount (range) can be specified. Fig. 5 shows an example IF statement. During deployment, an IF statement is fulfilled when at least one matching operation is found. THEN statements describe what operations to execute when all IF statements of a rule are fulfilled. They have only one sender, resource type, amount, and receiver. To allow the specification of dynamic rules, variables and wildcards can be used. A variable refers to a sender, receiver, resource type, or amount used in a previous statement. They are generated on the fly, for every statement. For example, in any THEN statement, one can refer to a receiver in any previous IF statement, by selecting "Receiver from IF Statement 'X'" (see Fig. 6). Additionally, certain wildcards can be used by the sender and receiver fields of IF statements, such as "All Organizations" or "Any Organization except 'X'". The latter allows to invert a selection, e.g., in a scenario where the participating organizations are "Store 1", "Store 2" and "Store 3", the possible sender(s) of an IF statement where the "except Store 1" was selected, would be "Store 2" and "Store 3". A combination of variables and wildcards is possible. The grammar for the rules is as follows:

```
<rule> ::= if(<ifstmts>) <thenstmts>
<ifstmts> ::= <ifstmt>; {<ifstmts>}
<thenstmts> ::= <thenstmt>; {<thenstmts>}

<ifstmt> ::= <operationtype>, ar(<amtrange>)
<thenstmt> ::= <operation>

<operationtype> ::= snd(<senders>), rt(<resoucetypes>), rcv(<receivers>)
<operation> ::= snd(<sender>), rt(<resoucetype>), a(<amt>), rcv(<receiver>)
<senders> ::= <orgs>|<custs>
<sender> ::= <org>|<cust>
<receivers> ::= <orgs>|<custs>
<receiver> ::= <org>|<cust>

<orgs> ::= <org> {<orgs>}
<custs> ::= <cust> {<custs>}
<rts> ::= <frts>|<nfrts>
<frts> ::= <frt> {<frts>}
<nfrts> ::= <nfrt> {<nfrts>}
<org> ::= ? all orgs from step 2 ?|<var>
<cust> ::= ? all custs from step 4 ?|<var>
<frt> ::= ? all FRTs from step 5 ?|<var>
<nfrt> ::= ? all FRTs from step 5 ?|<var>
<amtrange> ::= <amt>|<amt>-><amt>
<amt> ::= <number>|<var>|any
<var> ::= ? all generated variables on-the-fly ?
<number> ::= ? all existing numbers ?
```


IF

[+] Add IF Statement

Statement 1 (Delete)

Senders (*) ☐ Invert selection

Select: Any Organization | Any Customer

[O] Store 1 × [O] Store 2 ×

Choose all possible senders for this operation type.

Resource Types and Amount Range (*)

Select: Any Fungible RT | Any Non-Fungible RT

[FRT] Loyalty Coin ×

Choose all possible resource types for this operation type.

[1,100] ×

Enter a quantity such as 1,2,3 or amount range such as [1,20] (use $ for any)

Receivers (*) ☐ Invert selection

Select: Any Organization | Any Customer

[C] Any Customer ×

Choose all possible receivers for this operation type.

Fig. 5. User Interface: Step 7 - Define Rules - IF Statement

4.3 Layer 1 – Use Case Interaction

The “Use Case Interaction” component is a web app generated for a specified use case that allows organizations and end-users to consult their digital resources and transfer these resources amongst each other. The web application calls smart contracts functions, which result in transactions and state changes (such as balance of currencies or change of ownership of resources) on the blockchain.

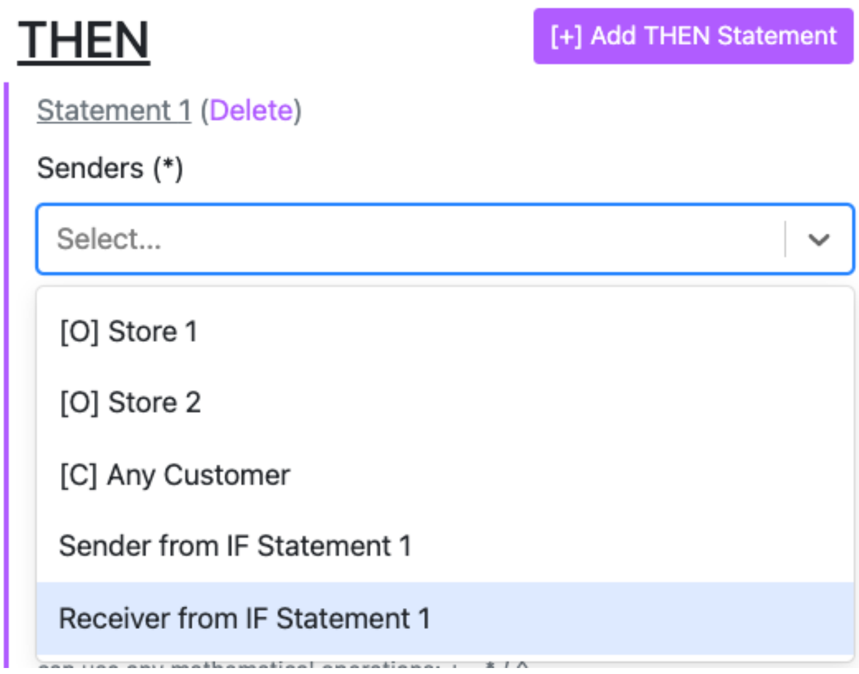


Fig. 6. User Interface: Step 7 - Define Rules - Use of Variables

4.4 Layer 2 – Smart Contract Generation

This layer generates the smart contracts necessary to deploy the use case, based on the information collected in the previous layer. As explained in section 2, it is important to consider smart contract size. Therefore, the generator distributes the logic across different smart contracts, including a controller, a contract for each token and each organization, a contract for the rules as well as the transactions, and a migrations contract. We have provided an implementation for the Solidity programming language, but one can easily add implementations for other smart contract languages.

We briefly explain below how the concepts from layer 1 are implemented in smart contracts. Note that in blockchains: (1) wallets are means used to identify individual end users (instead of traditional login and password), allowing them to hold resources, sign transactions and messages [29]; and (2) smart contracts are means used to facilitate interactions across wallets and other smart contracts.

Organizations and Owners. Typically, organizations are not controlled by a single person, but by multiple individuals instead. As explained earlier, we introduced organization owners for this. These should be able to sign transactions and messages in the name of the organization. Therefore, a contract holding the logic for the organizations and its owners will be generated, ensuring that the owners have the permission to execute transactions or sign messages on behalf of the organization.

End Users. Each end user is an individual who may hold, receive, or spend resources. Therefore, each end user will be represented by a wallet.

Resource Types and Resources. As the token standards indicate, a token specification on a blockchain is implemented by means of a smart contract [14, 15]. That contract can generate tokens compliant with their specification. As such, a resource type will be implemented by a token specification contract, while a resource will be an instance generated by such contract (represented by a token).

Property Types and Properties. Additional properties of a resource will be stored as fields or functions in the token contract corresponding with the resource type. As such, resources having properties in layer 1, correspond to tokens having field values or functions in layer 2.

Operation Types and Operations. Operations in layer 1 correspond to blockchain transactions in layer 2, and defined operation types correspond to allowed transactions on the blockchain. Transactions which do not correspond to a defined operation type will revert.

Rules. A rule will be implemented with one or more function(s) in Solidity.

4.5 Blockchain Network

The blockchain network will allow interactions with and deployment of smart contracts. The network of deployment can be chosen during use case setup.

4.6 Advantages

The structure of the framework, and the use of decoupled layers, has a couple of advantages. First, when organizations want to adapt a use case or migrate the use case to a different network, one may redeploy a new copy of the contracts with a simple click, since use case specifications are stored on a (centralized) backend. Note that this backend only stores the data needed to generate the contracts for the use case. The actual interaction of the end users with the generated contracts for the use case occurs on the (decentralized) blockchain, and the generated smart contracts are shown to the user before final deployment on the blockchain (to ensure transparency).

Also, as discussed above, Layer 2 is currently implemented to support the Solidity programming language. However, if another blockchain is desired, or if one wants to add the ability to map the use cases to a different program language (e.g. Rust[5] for Solana), the framework can be extended by adding a new implementation to Layer 2 to support that program language.

[5] https://docs.solana.com/developing/on-chain-programs/developing-rust

5 Use Case Specifications - Examples

We illustrate the process of specifying a use case by two use case examples. Note that use cases can be deployed on various blockchains, such as Ethereum, Binance Chain and Cardano. Each has its advantages and disadvantages, such as popularity (e.g. social community), trustworthiness (e.g. degree of decentralization), efficiency (e.g. transaction speed) and cost (e.g. transaction fees, gas price, ..). Proper consideration must be given to above factors before deploying the use case on a certain blockchain.

5.1 Loyalty Programs

In loyalty programs, stores offer rewards to customers as an incentive for their frequent or recurring purchases. Such programs are offered in different forms: loyalty points, punch cards, cash back programs, and more, as the analysis in [30] shows. In this use case example, we will consider loyalty points. In most current systems, the customer can only use its points in the same store or a store from the same business group, and does not hold true ownership of its points as explained in the introduction. Making use of the described framework, we can generate an application serving a loyalty program where the customer could for instance earn points in store A by buying water packs and spend these points in store B to get a free cold drink (instead of a water bottle from store A). For this, and depending on their business model, stores A and B will have to agree on a settlement contract. A possible option could be that store B needs to return 15% of the points the customer spent in its store, to store A, since the points rewarded to the customer were issued by store A. We now illustrate the specification of this use case. For this use case, steps 1 to 5 are obvious. In step 6, we define two operation types: (1) stores need to be able to award loyalty points to customers and (2) customers need to be able to spend them in stores.

Using the given grammar, these operation types are defined as follows:

```
<operationtype> -> snd(all_orgs), rt(loyalty_coin), rcv(all_custs)
<operationtype> -> snd(all_custs), rt(loyalty_coin), rcv(all_orgs)
```

In step 7, we define the 15% return rule. In natural language, this rule could be formulated as follows: “IF a store awards loyalty points to a customer AND that customer spends those points (same amount or less) in another store, THEN this second store will pay the first store 15% of the received points as compensation”. Using the grammar, the rule is as follows:

```
<rule> ->
      if(snd(store_a store_b), rt(loyalty_coin), rcv(any_customer), ar(any);
        snd(receiver_from_if_1), rt(rt_from_if_1),
        rcv(any_except_sender_from_if_1), ar(0->amt_from_if_1);)
      then(snd(receiver_from_if_2), rt(rt_from_if_2), a(amt_from_if_2*0.15),
           rcv(sender_from_if_1);)
```

5.2 University Collaboration

Most academic institutions store degrees issued to their students on their own and/or government-controlled, centralized infrastructure and provide a verified paper degree to the student. This does not give the student true ownership of the degree: in case of doubt, other institutions will verify the ownership of the degree by contacting the issuing institution. Making use of the proposed framework, a system can be developed that allows a student to prove to any party that he/she holds the degree. Also, constraints can be easily set up. As an example rule, one can specify that the student that wants to enroll in a university will only receive an enrollment certificate of the university when he/she has a secondary education certificate and he/she has deposited enough budget (e.g., expressed in “university coins”) to pay for the enrollment.

We now illustrate the specification of this use case in our framework. For this use case, steps 1 to 5 are obvious. In step 6, we define three operation types: (1) secondary schools should be able to award students with secondary education certificates; (2) universities should be able to award students with enrollment certificates and (3) students should be able to pay universities for enrollment:

```
<operationtype> -> snd(all_schools), rt(se_cert), rcv(all_students)
<operationtype> -> snd(all_univs), rt(enr_cert), rcv(all_students)
<operationtype> -> snd(all_students), rt(univ_coin), rcv(all_univs)
```

In step 7, we define a rule which requires students to pay (for the enrollment) (e.g., “10” university coins) before receiving their enrollment certificate. Moreover, the students should have received a secondary education certificate. In natural language: “IF any school awarded a secondary education certificate to a student AND that student will receives an enrollment certificate from a university, THEN that student needs to pay ‘10’ university coins” to this university:

```
<rule> -> if(snd(any_school), rt(se_cert), rcv(a_student), ar(1);
             snd(any_univ), rt(enr_cert),  rcv(receiver_from_if_1))
          then(snd(receiver_from_if_1), rt(university_coin), a(10),
               rcv(sender_from_if_2);)
```

6 User Study

A pilot user study has been performed for the Use Case Specification web application developed for Layer 1. The purpose of this first user study was to investigate the usability of the tool for the target audience (business-level people), as well as to evaluate the terminology and principles used. It concerned a formative evaluation [31], i.e., an evaluation conducted during development to investigate whether improvements to the product are needed.

At the time of writing, the pilot study was performed by one participant but in the coming time more people will be asked to participate. The participant was a male, 56 years old, studied Bachelor of Engineering, worked as an Engineer and managed a family-owned business. Currently, he is director of a company and an independent Salesman. This person did not have any knowledge of blockchain or smart contracts.

6.1 Setup of an Evaluation Session

We start with providing the participant with background information on the framework and its goal. It is explicitly mentioned that the goal of the user study is to evaluate the ease with which the tasks can be done with the web application's user interface by people without IT background. The participant is also informed that (s)he will be asked to fill in a questionnaire about the application that would cover the following topics: Attractiveness, Efficiency, Perspicuity, Dependability, Intuitive Use, Clarity, and Response Behavior. At any point during the study the participant can ask for clarifications or more information. Each evaluation session is performed under the supervision of the first author. The participant is encouraged to speak aloud while interacting with the system and is observed while performing the task, which is using the web application for specifying a use case that the participant can propose himself.

Next, a semi-structured interview takes place, involving filling in a questionnaire, as well as answering open questions.

As an introduction to the task, a demonstration of the user interface is given by browsing through the different steps of the application, giving the participant an idea on how to use it (without providing a detailed explanation). As a reference, we use and explain the "Loyalty Points" use case, mentioned in section 5.1. This takes about 30 minutes in total. Afterwards, we ask the participant to propose a use case (suitable for this framework) and specify this use case with the web application. The participant is guided, and explanations are given when needed.

For the questionnaire, UEQ+[6] is used, being a modular extension of the User Experience Questionnaire (UEQ). The following modules are included: Attractiveness, Efficiency, Perspicuity, Dependability, Intuitive Use, Clarity, and Response Behavior. In total, there are 5 questions with opposing pairs of product properties and 7 grades per pair for indicating the level of agreement with the terms. In UEQ+, the pairs are organized into groups related to one single aspect. For each group, the participant can state his or her opinion about the level of importance of the respective aspect. After having filled out the questionnaire, the participant is asked to clarify his/her answers.

Next, open questions regarding the usability of the web application are asked. The questions are about the naming conventions used, possible system improvements, intuitiveness, clarity of the application, clarity of the given oral introduction (explaining the system), and whether the participant would use such system in his/her business. For the questionnaire and the interview about 30 minutes in total is foreseen.

6.2 Results

The participant was able to finish the specification of the use case. Devising a use case took about one hour and entering its specification in the web application also took one hour. The use case he proposed was a mobility reward system, with the goal to make public transport cheaper and more accessible. A collaboration between stores and public transport, allows customers to earn mobility points in stores during shopping,

[6] https://ueqplus.ueq-research.org/

which they can use later as a payment method for public transport services, such as busses, trams, trains, public cycles, public e-steps, public cars and more.

According to the questionnaire, the participant experienced the user interface as: (1) quite attractive and pleasant in general, although the visibility and placement of some of the UI elements can be improved; (2) efficient and fast, though sometimes impractical (prefers more visual guidance); (3) overall perspicuous, though the learnability and ease of use could be improved by more visual guidance; (4) very dependable; (5) very intuitive; (6) organized, structured, ordered and well-grouped; (7) quite responsive; and the participant was very happily surprised by the ability to define "rules" in the user interface, since it allows to define settlements between businesses, without the need for trusted third parties (as in traditional businesses).

During the interview, the participant gave a more in-depth explanation of his opinion on certain aspects. The participant found the used naming conventions very clear, except for the terms "use case" and "mapper". By means of the oral introduction, the participant understood and could interpret all parts of the user interface. As a suggestion to improve the system, he suggested offering more guidance during the different steps, preferably by using a "pop-out effect"[7] as well as the ability to use existing monetary resources (e.g., such as Bitcoins or stablecoins like USDT[8]) in the rules section. He stated that all parts of the user interface were clear and well-organized, except for the specification of the "amount (range)" in the rules section (step 7), which was not very intuitive to him. He indicated that the system could definitely be useful for the participant's business.

During the performance of the task, we noted that the participant was often confused after creating an entity. It was not clear to him that sometimes multiple entities were needed (like for organizations, owners, operations, ...), and sometimes only one (like only one use case definition). This issue can be tackled by improving the guidance during the steps, as the participant suggested. In step 4, customers can be optionally defined. However, it was not clear to the participant that the definition of customers in step 4 is optional. This can be resolved by using the pop out effect.

6.3 Discussion

We can conclude that the participant in general had a good experience with the user interface, enjoyed the functionality, but lacked a more in-depth, preferably visual, step-per-step guidance. The provided explanation at the beginning of the session was required for the participant. This means that a small introduction should be provided in the tool. All the usability issues the participant had, can be resolved by adding more guidance to the web application, as well as by adding and/or tweaking UI components where needed.

According to this participant, the product is quite plausible, enjoyable, and has high potential. Of course, as this is only the opinion of one person, but the purpose of this evaluation was to verify the approach and decisions taken (i.e., a so-called formative evaluation). After improving the software, a user evaluation with more participants will take place in order to be able to generalize the results.

[7] Occurs when one object visually pops out from the rest of the objects.
[8] https://tether.to/

7 Conclusion and Future Work

We have presented a framework allowing the easy setup of systems for the exchange of resources between organizations and end users, in which the end users hold ownership of resources and can spend/use them across various organizations. We presented the principles used, as well as the layered architecture of the framework. The extensible character of the framework allows to add support for more implementation languages and networks.

The focus of the paper was on the specification layer that allows people without an IT background to specify their use case. This is supported by a web-based user interface. Other than specifying general information about the use case, the UI allows the user to express requirements and constraints by means of IF-THEN rules. We illustrated the specification by means of two use case examples.

Currently, the user interface is improved based on the feedback of the user study and a user evaluation with more participants is set up. Future work related to Layer 1 of the framework includes the development of a multi-user version (allowing multiple users to access the same use case), adding rules that allow the definition of resource types in statements based on individual properties, and using network native currencies (such as Ethereum) in the rule definitions.

References

1. Golosova, J., Romanovs, A.: The advantages and disadvantages of the blockchain technology. 2018 IEEE 6th Work. Adv. Information, Electron. Electr. Eng. AIEEE 2018 - Proc. (2018). https://doi.org/10.1109/AIEEE.2018.8592253.
2. Scherer, M.: Performance and Scalability of Blockchain Networks and Smart Contracts. (2017).
3. Underwood, S.: Blockchain beyond bitcoin. Commun. ACM. 59, 15–17 (2016). https://doi.org/10.1145/2994581.
4. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System - Whitepaper, https://bitcoin.org/bitcoin.pdf, last accessed 2021/07/21.
5. Christidis, K., Devetsikiotis, M.: Blockchains and Smart Contracts for the Internet of Things. IEEE Access. 4, 2292–2303 (2016). https://doi.org/10.1109/ACCESS.2016.2566339.
6. Giancaspro, M.: Is a 'smart contract' really a smart idea? Insights from a legal perspective. Comput. Law Secur. Rev. 33, 825–835 (2017). https://doi.org/10.1016/J.CLSR.2017.05.007.
7. Binance Smart Chain: A Parallel Binance Chain to Enable Smart Contracts Motivation.
8. Cardano | Home, https://cardano.org/, last accessed 2021/07/11.
9. Polkadot: Vision for a Heterogeneous Multi-Chain Framework, https://github.com/ethereum/wiki/wiki/Chain-Fibers-Redux, last accessed 2021/07/05.
10. Solana: Build crypto apps that scale | Scalable Blockchain Infrastructure: Billions of transactions & counting, https://solana.com/, last accessed 2022/03/31.
11. Pierro, G.A., Rocha, H.: The influence factors on ethereum transaction fees. Proc. - 2019 IEEE/ACM 2nd Int. Work. Emerg. Trends Softw. Eng. Blockchain, WETSEB 2019. 24–31 (2019). https://doi.org/10.1109/WETSEB.2019.00010.

12. Sweet Tools for Smart Contracts | Truffle Suite, https://www.trufflesuite.com/, last accessed 2021/07/11.
13. Embark into the Ether. | Embark, https://framework.embarklabs.io/, last accessed 2021/07/11.
14. Di Angelo, M., Salzer, G.: Tokens, Types, and Standards: Identification and Utilization in Ethereum. In: Proceedings - 2020 IEEE International Conference on Decentralized Applications and Infrastructures, DAPPS 2020. pp. 1–10. Institute of Electrical and Electronics Engineers Inc. (2020). https://doi.org/10.1109/DAPPS49028.2020.00001.
15. ERC | Ethereum Improvement Proposals, https://eips.ethereum.org/erc, last accessed 2021/07/12.
16. ERC-20 Token Standard | ethereum.org, https://ethereum.org/en/developers/docs/standards/tokens/erc-20/, last accessed 2021/07/12.
17. ERC-721 Non-Fungible Token Standard | ethereum.org, https://ethereum.org/en/developers/docs/standards/tokens/erc-721/, last accessed 2021/07/12.
18. EIP-1155: ERC-1155 Multi Token Standard, https://eips.ethereum.org/EIPS/eip-1155, last accessed 2021/07/12.
19. ERC20 Token Generator | Create ERC20 Token for FREE, https://vittominacori.github.io/erc20-generator/, last accessed 2021/07/05.
20. MetaMask, https://metamask.io/, last accessed 2021/07/05.
21. web3.js - Ethereum JavaScript API — web3.js 1.0.0 documentation, https://web3js.readthedocs.io/en/v1.3.4/, last accessed 2021/07/05.
22. BEP20 Token Generator | Create BEP20 Token for FREE on Binance Smart Chain, https://vittominacori.github.io/bep20-generator/, last accessed 2021/07/05.
23. OpenSea: Buy NFTs, Crypto Collectibles, CryptoKitties, Decentraland, and more on Ethereum, https://opensea.io/, last accessed 2021/07/05.
24. Astigarraga, T., Chen, X., Chen, Y., Gu, J., Hull, R., Jiao, L., Li, Y., Novotny, P.: Empowering Business-Level Blockchain Users with a Rules Framework for Smart Contracts. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics). 11236 LNCS, 111–128 (2018). https://doi.org/10.1007/978-3-030-03596-9_8.
25. López-Pintado, O., García-Bañuelos, L., Dumas, M., Weber, I., Ponomarev, A.: Caterpillar: A business process execution engine on the Ethereum blockchain. Softw. - Pract. Exp. 49, 1162–1193 (2019). https://doi.org/10.1002/SPE.2702.
26. Choudhury, O., Rudolph, N., Sylla, I., Fairoza, N., Das, A.: Auto-Generation of Smart Contracts from Domain-Specific Ontologies and Semantic Rules. Proc. - IEEE 2018 Int. Congr. Cybermatics 2018 IEEE Conf. Internet Things, Green Comput. Commun. Cyber, Phys. Soc. Comput. Smart Data, Blockchain, Comput. Inf. Technol. iThings/Gree. 963–970 (2018). https://doi.org/10.1109/CYBERMATICS_2018.2018.00183.
27. He, X., Qin, B., Zhu, Y., Chen, X., Liu, Y.: SPESC: A Specification Language for Smart Contracts. Proc. - Int. Comput. Softw. Appl. Conf. 1, 132–137 (2018). https://doi.org/10.1109/COMPSAC.2018.00025.
28. Ben Slama Souei, W., El Hog, C., Sliman, L., Ben Djemaa, R., Ben Amor, I.A.: Towards a Uniform Description Language for Smart Contract. 57–62 (2022). https://doi.org/10.1109/WETICE53228.2021.00022.
29. Mahmoud, Q.H., Lescisin, M., AlTaei, M.: Research challenges and opportunities in blockchain and cryptocurrencies. Internet Technol. Lett. 2, e93 (2019). https://doi.org/10.1002/ITL2.93.

30. Agrawal, D., Jureczek, N., Gopalakrishnan, G., Guzman, M.N., Mcdonald, M., Kim, H.: Loyalty Points on the Blockchain. Bus. Manag. Stud. 4, (2018). https://doi.org/10.11114/bms.v4i3.3523.
31. Redish, J. (Ginny), Bias, R.G., Bailey, R., Molich, R., Dumas, J., Spool, J.M.: Usability in practice. 885 (2002). https://doi.org/10.1145/506443.506647.