Software Engineering for DApp Smart Contracts managing workers Contracts

Giorgia Lallai[1], Andrea Pinna[2], Michele Marchesi[1], and Roberto Tonelli[1]

[1] Department of Mathematics and Computer Science - University of Cagliari
giorgia.lallai@hotmail.it,marchesi@unica.it, roberto.tonelli@dsf.unica.it
[2] Department of Electrical and Electronic Engineering (DIEE)- University of Cagliari
a.pinna@diee.unica.it

Abstract

We present an application of the BOSE and ABCDE development methodology to build a DApp system for managing real world contracts for temporary workers so that, *by design*, agreements, commitments and rules are respected for the specific domain and employment sector and so that employers and employees are safeguarded by design. This includes the possibility to provide access from public regulatory bodies to all the information and employment history. Ethereum Solidity Smart Contracts are designed to manage all the steps and to keep track of all Commitments and Agreements, of Employers and Employees and of job history. We built a working prototype of application where the system management is automated by mean of an easy to use web interface acting as a front-end interacting with the blockchain back-end.

1 Introduction

Blockchain applications are blooming in the most disparate sectors aside from the cryptocurrency original start. The introduction of the Ethereum platform solicited developers to produce decentralized applications that nowadays constitute the Ethereum DApp ecosystem. Despite such exponential increase of interest most DApp still lack of organized software development methodologies and even when the domain of application may seem interesting such un-organized DApp development does not guaranty security in sensitive applications neither grants that constraints and rules from the application domain are respected.

This is especially critical when Smart Contracts are meant to substantiate real life contracts, such as in the domain of employment. Contractors who want to recur to automated and self enforcing Smart Contracts to respect agreements and to fulfill commitments must be secure by design that some domain specific constraints are respected and the software code does not fail to fulfill the agreed rules.

In this work we present a case study where temporary employments are managed by a blockchain software system designed according to the Blockchain Oriented Software Engineering (BOSE) [7] approach to deliver blockchain software systems and using the ABCDE methodology [3] to identify actors, user stories, activity and sequence diagrams in order to provide the DApp system architecture with the desired properties *by design*.

In this specific case study the advantages of using a blockchain DApp system rather than a traditional software system are multiples.

First, the nature of the relationships between the employer and the employees in the specific case of temporary workers is controversial. In fact the first usually wants to optimize earnings and minimize costs, and usually the power of bargaining between the parties is unbalanced,

since the the temporary workers need to quickly find a job and are exposed to a competition with offers to downside for the salary in order to get the job. Using blockchain all data are recorded and accessible to anyone so that any offer to downside became transparent. Workers have access to information regarding average salaries and can ask for a pay comparable to the average. Public bodies can access to salaries and can verify if minimal Union salaries rules are fulfilled in the agreements.

Second, given the unbalance between each single temporary worker and a big company, the latter can take advantage of its position of power to delay or reduce payments, counting on the low probability of being suited from a temporary worker. Using Smart Contracts payments amounts and deadlines are automatically respected by the execution of the smart contract code.

Third, the employer can automate checks and steps needed to pay the workers and by automatic execution of smart contract code can verify if the workers completely fulfilled the agreed duties before any payment is executed.

Fourth, and most important, in this case study the application of BOSE and ABCDE methodology helps to simplify the system and the smart contract configuration and to clarify to all involved figures the actor roles and duties, so that anyone can easily understand conditions, agreements and duties.

We do not further bother the reader describing and discussing here the other well known and general advantages carried on by the use of a blockchain system in this specific case such as immutability, privacy, traceability and so on, that are easily understood for a blockchain system. For a comprehensive discussion about these aspects we remand to the conceptual proposal presented in [6]

2 The Employment Eco-system

The target of our design is to simplify the temporary employment procedure, to provide traceability of the procedure, to protect both employer and employee against misbehavior, to prevent out of law agreements, to provide public bodies devoted to controls and checks with an easy to use system, to render public and secure the ecosystem, to provide an easy to use and practical interface for the interaction between employer and employees.

Most importantly, we apply the BOSE and ABCDE methodology to devise the overall system's architecture showing how these approaches help to define actores, roles, constraints, functionalities and requirements for a specific domain.

The ABCDE methodology starts with the subdivision of the system in out-of-chain and in-chain components using the diagrams as prescribed by BOSE. This requires also to identify the system's actors.

We design the system so that there are two actors typologies, three human actors and two actors which are system components, for a total of five main actors.

The first typology, the human actors, are:

- *The employer*: it creates the work activity, it announces the request for one or more workers, it describes the job, the pay, the duties, the worker features, the time period for the request, and so on.
- *The worker*: it typically applies for a job, provides his CV, if possible examines and chooses among different job offers.
- *The work inspector*: he can access the employer's data and check how many hours he has registered for each worker. He will be able to compare these data with the results of any

workplace inspection.

The second typology are system components, in and out of chain:

- *The web platform*: a simplified web platform with an interface allowing to post new job offers, to insert job candidacies, to access information about the posted jobs.
- *The blockchain infrastructure*: it records smart contracts and transactions for the various job contracts, it allows to manage direct payments, it grants security and privacy

The blockchain infrastructure has the role of master ledger but also the role of protecting both parties against misbehavior and scams. In order to make concrete our system we used the most popular blockchain at the moment, namely the Ethereum blockchain, since it provides a nice environment to write smart contracts, compilers and debuggers are available, testnets for trials are free and working properly, and finally a complete environment for interacting with the blockchain from a web interface is provided by the web3.js library. Ethereum in fact provides a well tested ecosystem where many examples are available to use standard procedures and datatypes for developing decentralized applications. Ethereum smart contracts, implemented in Solidity, are well suited for the application of BOSE and ABCDE methodology and procedures for data recording and transaction managing are easily implemented. Furthermore the Ether cryptocurrency holds a true value in the market, since exchanges deal it in large amounts aganst fiat currency, and so it can be used to implement real money transfers and payments. In this specific contest public regulatory bodies are usually involved but nowadays the market of temporary jobs is setting the traditional system into a sort of crisis, especially because multinational firms can hire workers behind the protection of different legislative systems where regulatory bodies find it hard to intervene. The blockchain can be an alternative where contract conditions are directly guaranteed by the technology itself.

3 The model architecture and the Scenario

The initial system state is idle. This means that the system still has to receive an input deposit in order to create a new job offer from an employer which initially does not contain any information about the employer or the employee. This deposit will grant the workers that there is some initial amount of money to pay for the job. This must be considered as a set up before starting.

The first event is the creation of a new job offer where the employer uses the webApp to complete the procedure for inserting the offer with all the information needed. Each job offer has several properties, such as: a name, an id, a description, the working hours, the pay, the deadline. All the transitions could be represented by a state diagram according to UML representation which can be adapted to the blockchain specific features.

After this step the employer can create the announcement where the salary is limited to a maximum according to what deposited. At the end of the procedure the system automatically creates and sends to the blockchain various messages which include all the information needed to create the set of smart contracts for managing the job, this will be discussed later.

After created the offer the system's state changes passing to a waiting state for the candidates workers. The system is configured in order to accept, at this point, the workers applications for the job. Each worker sends a message to a specific Smart Contract for registering his/her candidature. Once a candidate is selected for a job the *hiring event* is launched. The employer obtains the id and the data of the applicant and sends a message to the blockchain to announce

the start of the working period. From this moment on, the worker can verify its working situation: the worked hours can automatically be certified according to different schemes and agreements but in any case the employer must certify the worked hours with a message sent to the appropriate smart contract. Once the amount of agreed working hours is reached the smart contract ends the job and sends the pay. During this event the system moves the salary deposited at the beginning to the employee account concluding the relationship. The worker, since everything is recorded, will be able to recover all the data history for any job performed.

3.1 Smart Contracts Design

We implemented two Smart Contracts typologies using the ERC721 token that we call *JobOfferManager* and *Employment* which are already registered and ready in the webApp. The platform can customize the Smart contracts by the data inserted from the employer according to the previous section. Once the event of creation of a new job offer is launched the two Smart Contracts are created, configured according to the information inserted by the employer, and deployed into the blockchain, linked to each other. The JobOfferManager implements various functions, such as the insertion of the deposit in ETH, the creation of a new job offer, the hiring and the payment. The Employment Smart Contracts contains all the information related to candidates. It also allows to the employer to increase and certify the working hours as worked by the employee, to end the job once the working hours are completed. The worker can apply to different job offers, withdraw a candidacy, send a request for a job. The features of the two Smart Contracts are described by user stories which are reported as a list or according to the diagram shown in fig. 1

The user stories are:

- ETH deposit: the employer put a deposit into the contract
- Announce: the employer sets a new job offer with description and features.
- Candidacy: workers send a job application to non expired offers
- Candidacy Withdraw: a worker can withdraw a candidacy
- Visualization: the employer can examine the candidacies
- Hiring: the employer selects a candidate and hires him/her.
- Hour registering request: the employee send a request to the employer to record and certify all worked hours up to that day.
- Confirm: the employer examines the request and can decide to asseverate the worked hours.
- Announcements Visualization: the employer visualizes the job offers he/she created
- Candidacy Visualization: the applicant visualizes the job offers he/she applied to
- Work Visualization: the employee visualizes the job he/she is working on.
- Career: the worker visualizes his/her work history
- Offers Visualization: any user visualizes all job offers (expired or active)
- Expired Offers: any user visualizes the expired job offers

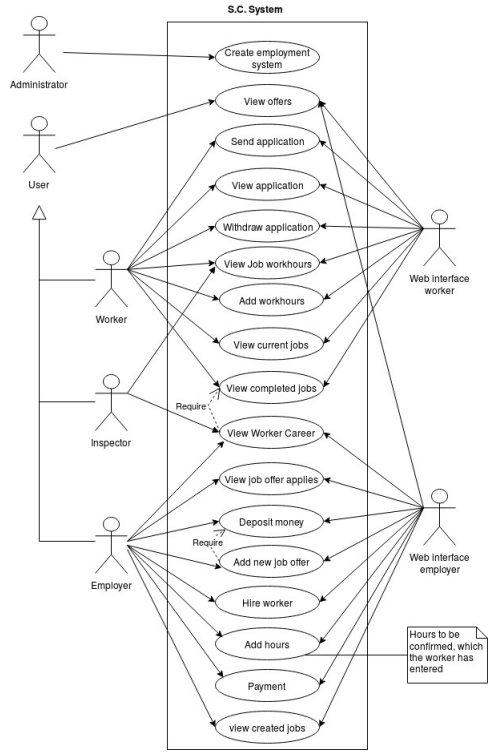


Figure 1: User Stories diagram for the Smart Contract implementation.

- Current Offers Visualization: any user visualizes all job offers not yet expired

For each user story we implemented a corresponding web page in the prototype.

Once the use case diagram for actors has been designed it is easy to proceed to the design of the Smart Contracts. According to what reported above, we implemented into the JobOfferManager Smart Contracts the following state variables:

- address owner: address of contract's creator - uint32 lastid: token holding the number of created offers - struct jobOffer (data representing one job offer)

- uint256 expirationDate: the expiration day
- address payable worker: Ethereum employee's account address
- address employer: Ethereum employer's account address
- string name: offer's name
- string info: offer's description
- uint8 workhours: working hours to be worked
- uint salary: salary offered

- *struct Jobs*:
– uint32[] jobs: array holding all the job offers created

- struct OnGoingJobs – uint32[] onGoingJobs: array holding all the jobs a specific worker is working on at the moment

- mapping(address => uint256) internal depositOf : mapping returning the ETH deposited in the Smart Contract by a specific address (associated to an employer)

- mapping(address => Jobs) internal offersBy: mapping returning the job offers created by a specific address

- mapping(address => OnGoingJobs) private hiredinjobs: mapping associating the employee address to all the job he/she is actually working on

- mapping(uint32 => jobOffer) private jobs: mapping returning the description of a job offer given the offer id

- mapping(uint32 => bool) public moneyIsReturn: mapping returning a boolean stating if the deposit has been withdrawn given an offer id.

The ABI interface is reported below according to the user stories diagram devised above.

-Constructor

-getNumberOfOffers(): outputs the number of offers created

-getName(uint32): outputs the offer's name

-getExpirationDate(uint32): outputs the expiration date

-getSalary(uint32): outputs the offer's salary

-getAddressWorker (uint32): outputs the employee's address

-getAddressEmployer (uint32): outputs the employer's address

-getInfo(uint32): outputs the offer's description

-getAmountHours(uint32): outputs the number of minutes to be worked for a job

-getJobOffer(uint32): outputs all the features of a job offer

-getArrayActiveOffer (): outputs all non expired offers

-getDepositedAmount(): outputs the amount deposited in the contract from a given address

-getOffersBy (address): outputs the offers created by a give address

-getApplicantOf (address): outputs an array containing all the jobs a worker is hired for

-getTokenId(): outputs the token's actual value

-getBalance(): outputs the amount deposited in wei

-getIsActiveOffer(uint32): verifies if an offer is expired or not

-getIsMoneyIsReturn (uint32): outputs if, given a job offer, the deposit has been withdrawn

-newJob(uint256, string memory, string memory, uint8, uint): creates a new job offer with all the details

-hireWorker(address payable, uint32): starts the work relationship. It is called by the employer to hire the employee for a given job offer

-payment(uint32): once the job duties are completed this function is called by the employer to pay the employee.

-moneyReturnsEemployer(uint32): it reimburses the employer once a job offer has expired without any worker hired.

Next we report as well the state variables for the *Employment* Smart Contract:

-address owner: address of contract's creator

-address payable scJobOfferManager: address referring to the Smart Contract managing the job offers

-*struct Applicant*:

——address[] applicant: array containing all candidacies

-*struct JobDone*

——uint32[] jobsDone: array containing all the jobs done by a worker

-*struct RequestHours* (data realted to the hours requested by a worker)

—- uint32[] idOffer: array containing all id of job offers without requests of adding hours

—- uint[] numberHours: array containing the number of requested work hours

-mapping (uint32 => uint) internal workhours: mapping providing the number of hours worked by an employee given the offer id

-mapping (uint32 => Applicant) internal applicantsOf: mapping providing the candidates of an offer given its id

-mapping (uint32 => uint) internal requestHours: mapping providing the number of hours requested for being added given the offer id

-mapping (address => RequestHours) internal requestHoursForEmployer: mapping providing the offer id and the number of requested hours given the employer's address

-mapping (address => JobDone) internal jobsDone: mapping providing the work concluded given the worker's address

Finally we report the ABI Interface for the Smart Contract *Employment*

-Constructor

-setJobOfferAddress(address payable): the function for setting the address of JobOfferManager Smart Contract

-getJobOfferAddress(): outputs the address of JobOfferManager contract

-getIsSetJobOfferAddress(): returns if the address of JobOfferManager contract has been set or not

-getIsEqualToJobOfferAddress(address payable): verifies if the value of variable scJobOfferManager matches the address of the contract provided in input (the contract Employment can refer only to one single contract managing the job offers)

-getApplicantOf(uint32): outputs the array containing candidates for an offer

-getJobsDone(): outputs the array containing all concluded jobs

-getRequestHours(uint32): outputs the number of hours required for an offer

-getRequestHoursForEmplyer(): outputs two arrays containing the offers' ids and the number of hours requested by each offer

-getHoursDone(uint32): outputs the number of hours worked up to that moment by an employee

-getHourMissing(unit32): outputs the number of remaining working hours to be completed

-jobCompleted(uint32): verifies if the work has been executed and performs the payment

-addWorkdays(uint32, uint): it is called by the employer to update the hours worked by an employee and activates the payment procedure (calling jobCompleted) once if the number of working hours agreed has been reached

-requestAdditionalHours(uint32, uint): is called by the employee to ask for adding the worked hours

-workerApplies(uint32): called by a user to apply for a job offer

-withdrawCandidacy(uint32): called by a user to withdraw from a job offer

3.2 Contracts Diagrams

Solidity allows to define Smart Contract in a way very similar to how classes are defined in Object Oriented (OO) programming languages. It supports inheritance, it includes a Constructor to deploy the Smart Contract on the blockchain, it supports interactions among contracts by means of transactions messages, so that a contract can call and activate functions of another

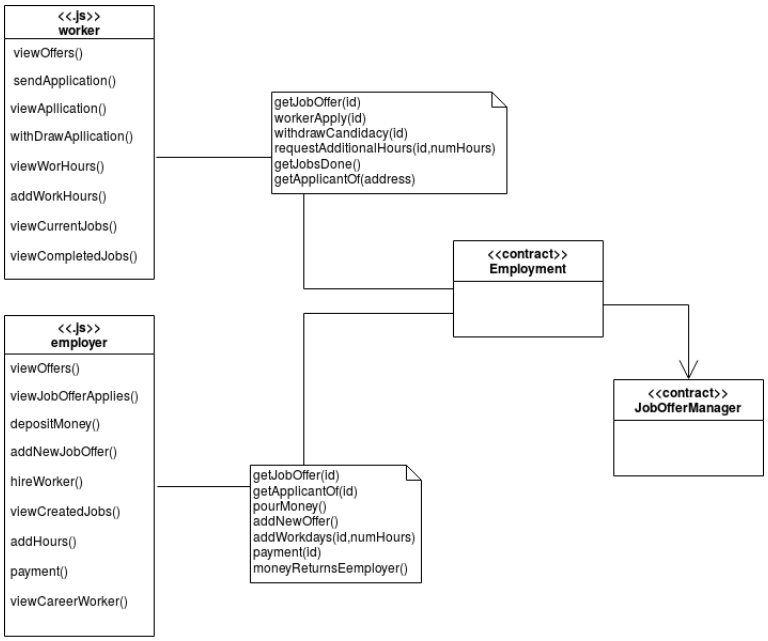


Figure 2: Smart Contract - Class diagram for the DApp implementation.

one, it supports events managing. According to BOSE and ABCDE methodology it's convenient to develop diagrams for Smart Contracts design and interaction which adopt the same schemes used in OO Software Engineering such as UML diagrams. We already presented the diagrams for Blockchain Software Engineering for users stories and for state diagrams in the previous section. In this section we describe the devised Smart Contracts and their interaction using the Contracts Diagram. Figures 2, 3 and 4 show the Smart Contracts diagrams and the relationships among them. These are described as in the usual UML diagrams, with similar meanings and with the use of stereotypes to characterize specific aspects of blockchain software. In particular the use of data structures and of libraries is easily described and allows for a clear and fast development of the smart contract software code in Solidity acting as a guide for developers. Fig. 2 also shows how it is possible to represent the interaction among in-chain and out-of-chain components.

3.3 Activity Diagram

In order to define and analyze the dynamic behavior of the system we designed the activity diagrams for the employer and for the employee. This is particularly important in Blockchain Software Engineering since the activity flow correspond to dynamic behaviours on the blockchain as well for the messages flow. In fact activities can match transactions in the blockchain which may occur between smart contracts. On the other hand, for activities that are performed internally to a single smart contract there is the possibility of generating the events related to these activities (this is typical in solidity). Thus in the design phase the activity diagrams provide a useful analysis tool to understand which operations need to be executed by means of blockchain transactions and which operations need to be recorded and tracked by means of events. The various activities in the diagram in general correspond to code execution or message

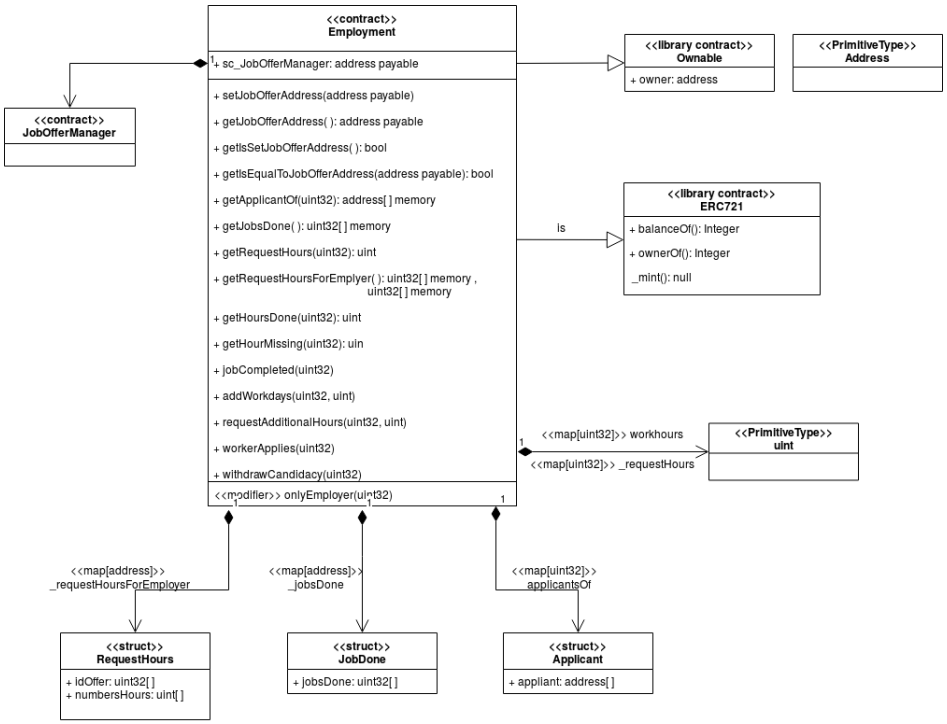


Figure 3: Smart Contract Class diagram for the *Employer* Smart Contract.

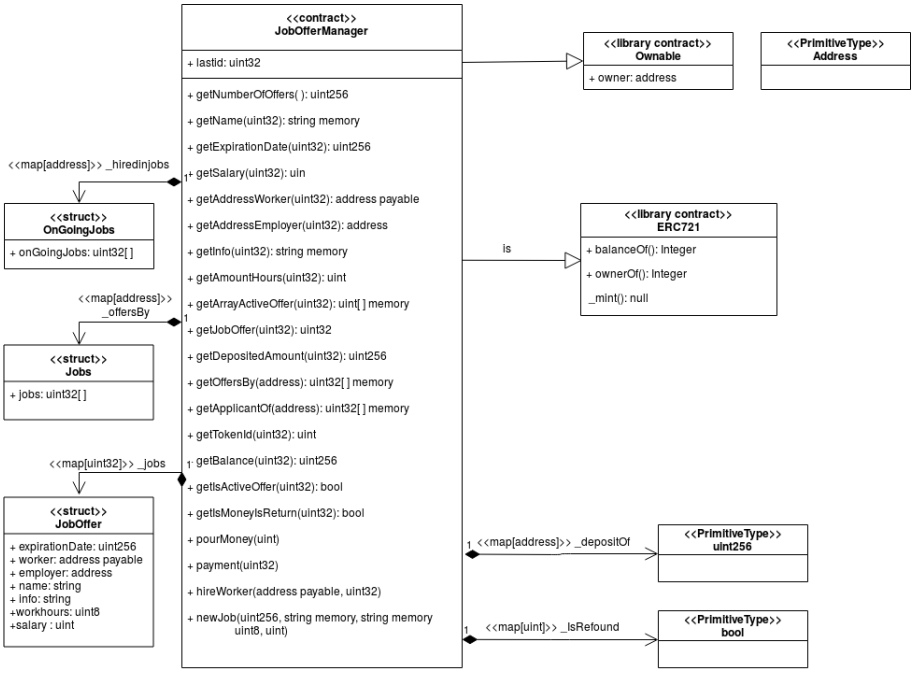
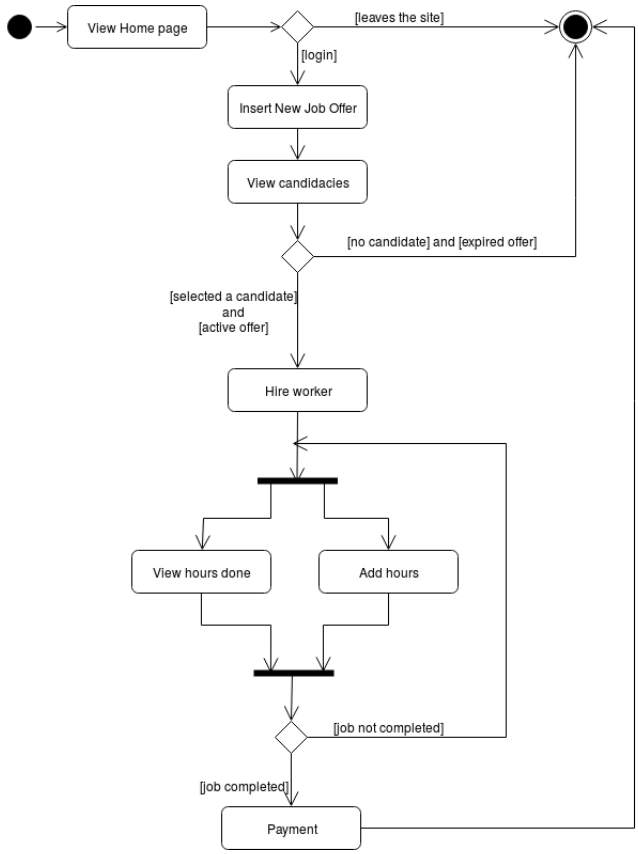


Figure 4: Smart Contract Class diagram for the *JOBOfferManager* Smart Contract.



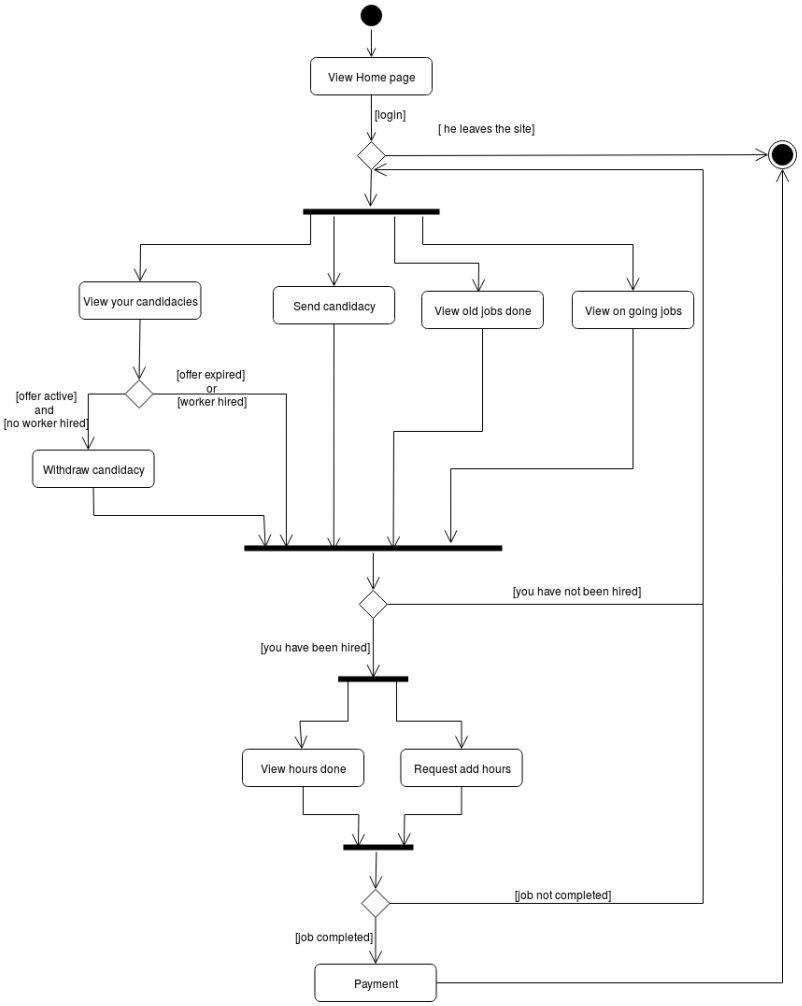
employer.png

Figure 5: Activity diagram for the *Employer*.

calls. In figures 5 6 we report the activity diagram for the employer and for the employee. The diagrams provide useful insights for identifying blocks and pieces of working software code for the in and out of blockchain components.

3.4 Sequence diagram

In the diagram 7 we represent the sequence of operations that the actors and the system perform to add work hours for a given job until the completion of the agreed number of hours. The worker accesses the workers' web interface and adds work hours to a job. He provides the job id. The web interface collects the request and calls the function "requestAdditionalHours (id, n)" of the employment smart contract which verifies that the worker (the sender of the request) is actually registered as a worker of the given job (with a given id) and the job is active. If so, it adds the N work hours as requested. The employer can check the amount of work hours to be confirmed calling the function viewHours() of the employers' web interface. The web interface loops in the job created by the employer and requests the employment smart contract to return the work hours waiting to be confirmed. Once data are collected, the web interface shows the list to the employer. The employer can now certify the work hours of a worker, given a job id. He accesses the employers' web interface and calls the function "selectJobOffer" to request



worker.png

Figure 6: Activity diagram for the *Employee*.

the validation of the requested work hours. The interface calls the function "addWorkdays" for a given id and an amount N. If the id is valid, the work hours are validated. Now, if the contractual limit of work hours is reached, the system calls the jobCompleted function which runs the "payment" function of the jobOfferManager contract.

4 The Prototype

Based on the application of all principles and methods adopted from BOSE and ABCDE methodologies we developed the Solidity code for the Smart Contracts (not reported here) and built the DApp system which provides the users with a user friendly web interface enabling the implementation of all the features described. In fig. 8 we report as an example the web interface providing the functionality for the insertion of a new job offer by the employer. The web interface uses "metamask", a bridge to run Ethereum DApps right in your browser with-

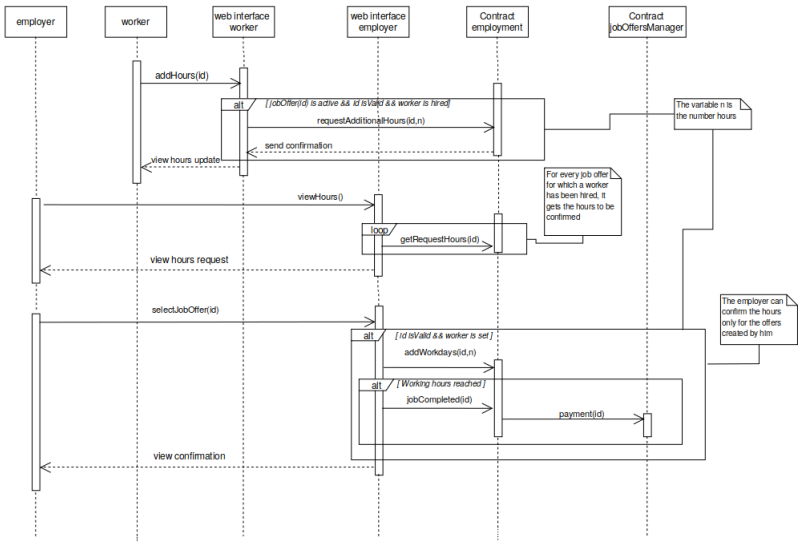


Figure 7: Sequence diagram of the hours registration and of the payment phases.

out running a full Ethereum node, for providing the communication channel between DApp and blockchain. The web interface allows the various actors of the system to interact with the blockchain through the DApp so that no actor needs to understand or to know the working principles of the blockchain system. Every feature is provided by a user friendly web page were input and output data can be inserted and read and events and function calls can be managed. We devised various interfaces to provide specific features depending on the users roles with menus adapted to the different roles. We deployed the Smart Contracts on the Ropsten test net in order to test our prototype under all working conditions.

5 Related works

In recent years, the research on the blockchain application in hiring management and work management is slowly expanding. In 2018, Oink et al. [4] proposed a blockchain based system to avoid errors in the recruitment phase of industrial personnel and in the management of human resources. That system uses a blockchain architecture to store, validate and manage workers' information. In this way, highly interconnected industries (i.e. 4.0 industry) and smart cities can share trusty data.

More recently, Peisl and Shah [5] evaluated the potentiality and the impact of the block chain technology applied to the work management, focusing on the employment life cycle and analyzing a set of use cases. They remark need of further study. For instance, current studies include literature reviews on the use of the blockchain technology in work contexts [1].

The design of blockchain based system for recruitment is the topic of the work of Won-Yong and Min [2] which focus on workers' skill certification, that helps the recruiters to chose the best applicant of a given job. From the other point of view, Gandhi et al. presented the implementation of a decentralized system to improve transparency of contractual conditions in the freelancer working relationships.

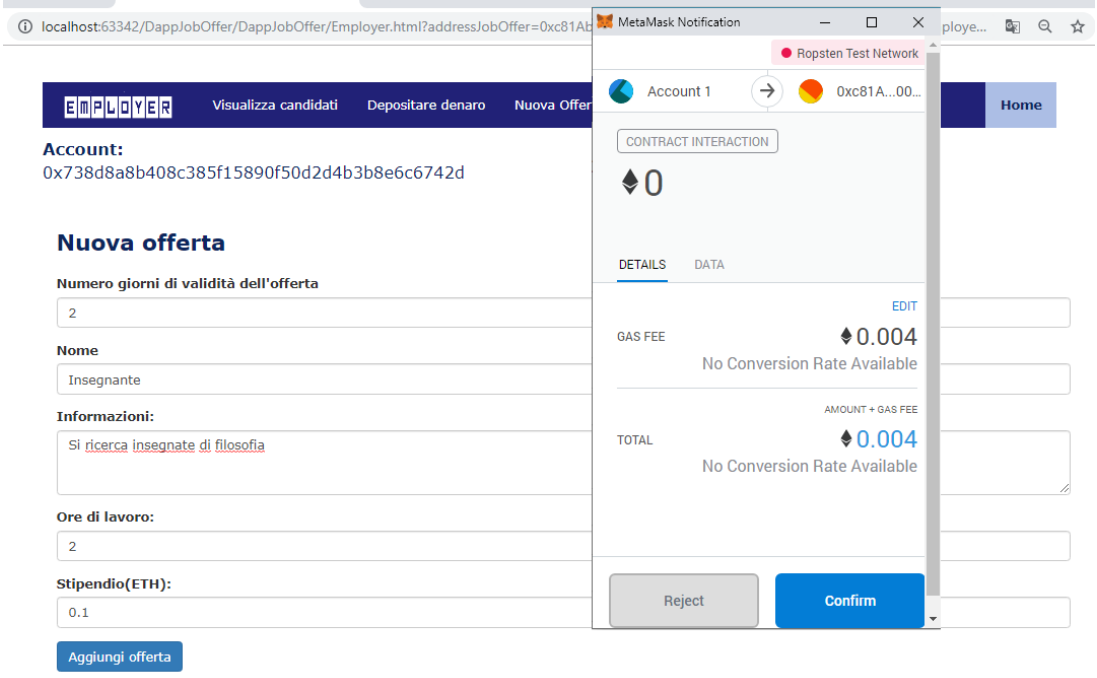


Figure 8: Web interface developed for the interaction with the DApp system.

All these works present two weakness. They focus only on the creation of the contractual relationship and does not include a specific automation in payment when work is completed nor provide additional protection to the worker or they do it only marginally.

6 Conclusions

In this case study we applied the BOSE and ABCDE methodology to devise a DApp for managing temporary employments so that by design the employers and the employees are able to easily identify roles, constraints, commitments in the specific domain. This approach allows to build a DApp software product in which all the requirements and features are determined and recovered by the diagrams adopted by the methodology so that Smart Contracts variables and ABI are quickly and precisely identified. The approach reduces risks of failure since in-chain and out-of-chain components are identified by design since the very beginning, and smart contract structure and interactions are well defined before the software development. We show how the approach successfully guided us to produce a working prototype for managing the case study of the temporary employments.

References

[1] Amer A Hijazi, Srinath Perera, Ali Alashwal, and Rodrigo N Calheiros. Blockchain adoption in construction supply chain: A review of studies across multiple sectors.

[2] Won-Yong Jeong and Min Choi. Design of recruitment management platform using digital certificate on blockchain. *Journal of Information Processing Systems*, 15(3), 2019.

[3] Michele Marchesi, Lodovica Marchesi, and Roberto Tonelli. An agile software engineering method to design blockchain applications. 14th Central and Eastern European Software Engineering Conference Russia, page 3. ACM, 2018.

[4] MH Onik, Mahdi H Miraz, and Chul-Soo Kim. A recruitment and human resource management technique using blockchain technology for industry 4.0. 2018.

[5] Thomas Peisl and Bahadur Shah. The impact of blockchain technologies on recruitment influencing the employee lifecycle. In Alastair Walker, Rory V. O'Connor, and Richard Messnarz, editors, *Systems, Software and Services Process Improvement*, pages 695–705, Cham, 2019. Springer International Publishing.

[6] Andrea Pinna and Simona Ibba. A blockchain-based decentralized system for proper handling of temporary employment contracts. In Kohei Arai, Supriya Kapoor, and Rahul Bhatia, editors, *Intelligent Computing, SAI*, pages 1231–1243, Cham, 2018. Springer International Publishing.

[7] S. Porru, A. Pinna, M. Marchesi, and R. Tonelli. Blockchain-oriented software engineering: Challenges and new directions. 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), pages 169–171, May 2017.