Blockchain's fame reaches the execution of personalized touristic itineraries

Amina Brahem, Nizar Messai, Yacine Sam, Sami Bhiri, Thomas Devogele, Walid Gaaloul

Blockchain's fame reaches the execution of personalized touristic itineraries

Amina Brahem*[1,2], Nizar Messai†[1], Yacine Sam‡[1], Sami Bhiri§[2], Thomas Devogele¶[1], Walid Gaaloul‖[3]
Email: * brahem.amina@gmail.com, †‡¶firstname.lastname@univ-tours.fr, §sami.bhiri@gmail.com, ‖ walid.gaaloul@telecom-sudparis.eu
[1]*LIFAT, University of Tours, Tours, France*
[2]*OASIS, National Engineering School of Tunis, University Tunis El Manar, Tunis, Tunisia*
[3]*Telecom SudParis, UMR 5157 Samovar, University Paris-Saclay, Paris, France*

***Abstract*—Blockchain trends cover more and more tech domains making it one of the most used technologies in the last few years. This is due to two essential aspects. First, it is a distributed peer-to-peer network where there is no need for third part to execute operations between peers. Second, blockchain implements mechanisms to make the most data sensitive operations executed in a trusted way. Regarding these attractive aspects, we intend in this work to use the blockchain technology for the implementation of touristic itineraries. We consider the latter as process choreographies involving different participants. We intend to model and execute the touristic itineraries generated from a personalized trip planner called CART in raw XML format in a way that respects this collaborative aspect and resolves the problem of trust. We will propose a pattern and its transformation rules to reconstruct the itineraries presented in declarative annotations such as XML to smart contracts written in some smart contract-specific programming language called Solidity. Experimental results show promising perspectives of the deployment of the proposed solution to execute touristic plans.**

***Index Terms*—Process choreography, touristic itinerary, blockchain, smart contract, declarative programming.**

I. Introduction

In a digital and interconnected world where exchanges occur in distributed networks, a need for mechanisms to ensure trust between different involved parts becomes primary. Touristic trip planning systems take in multiple participants like the service providers which compete and/or cooperate to fulfill a given request. The latter has as parameters the tourist constraints such as time and budget limitation, maximum number of parking spots, etc. The response is naturally the touristic trip plan. Due to customization aspect and increasing interaction with connected devices, the composition of touristic itineraries process is difficult to structure as user requirements change and so does the data services selected to construct them. CART (Configured mAshup Recommender application for personalized Trip planning) [1], [3], as an itinerary planning system, overcomes this problem by using a data mashup based approach [1] to aggregate data collected from different web resources in order to offer a set of personalized trip plans. Logically, the deployment and execution of touristic itineraries, which represent one of the main contributions in this work, will not be based on an orchestration where a single entity has full control and knowledge about all involved participants and tasks. We will soundly model a trip plan as a process choreography taking into consideration concurrence/collaboration between different engaged partners who do not know neither trust each other. Basically, for the case of touristic plans monetary exchanges like buying museum passes or paying for parking may be executed which raises concern about trust.

The emerging blockchain technology [7], [9], [12], [13] and especially the second generation of blockchain and its first class citizens the smart contracts, comes as a promising solution to execute collaborations of business processes and to overcome the problem of trust. Solutions like Caterpillar [11] and Lorikeet [15] already exist. However, they force the collaborative process to be modeled in Business Process Model Notation (BPMN), as structured business processes. In the present work, we intend to: (i) model touristic itineraries shaped from a data-mashup approach in the CART system as process choreograpahies – itineraries are generated from CART in raw XML format ; then (ii) deploy and execute the choreographies in the blockchain. To do so, we will propose a pattern and its transformation rules to reconstruct the itineraries presented in declarative annotations such as XML to smart contracts written in some smart contract-specific programming language called Solidity [6]. Solidity smart contracts are programmable contracts that execute on top of the Ethereum [6] blockchain one of the most famous blockchain platforms.

The reminder of this paper is structured as follows: background is presented in Section 2. Section 3 gives an overview of the related work. In Section 4, we present our pattern and transformation rules of touristic itinerary viewed as a business process choreography from XML to Solidity smart contracts. Section 5 includes a running example and some implementation details to execute the generated smart contracts. Section 6 concludes the paper and highlights future research directions.

II. Background

A. Touristic itinerary planning

Consulting sites like TripAdvisor, Yahoo Travel or Lonely Planet is a very common behavior of a tourist who wants to discover the more interesting Points of Interest (PoIs) to visit (museum, restaurant, hotel, etc) and plan for his trip. In this regard, PoI recommending systems also called trip or itinerary

planning systems offer an interesting solution for the automatic construction of a tourist's itinerary. The tourist has to only give his preferences and constraints like the total duration of the visit, the budget's limit or the maximum number of parking lots. The existing solutions compete to offer the most "personalized" itinerary like CART. CART models the trip planning problem as configuration of composite services [3]. Authors adopt an interactive and incremental data mashup process applied on data extracted from multiple web services to deliver a set of itineraries that respond the best to the user's preferences. CART generates touristic plans or itineraries that we will take as input in our work.

B. Blockchain and smart contracts

A blockchain [4] is one type of a distributed ledger where data is not only distributed but organized in sequence of blocks. The integrity of the data is ensured by cryptographic techniques. A block is a container data structure. It is composed of a header and a list of transactions. The blocks are chained in an append way where each block's header includes a hash of the block's transactions, as well as a copy of the hash of the prior block's header. The blockchain network, maintained by independent computers referred to as nodes or peers to record, share and synchronize transactions in their respective electronic ledgers. Peers do not know or trust each other but can connect and cooperate to validate operations executed on the blockchain. A transaction is a signed and identifiable piece of data that authorizes the sending of funds from one account to another, stores parameters such as monetary value in case of Bitcoin where only monetary operations (sending and receiving bitcoins) are allowed. We can trust the entire system of the blockchain because, first, everyone in the blockchain keeps a (partial or full) copy of the chain and can check what is exactly happening in the blockchain. Furthermore, transactions on the blockchain are agreed i.e., a transaction is accepted when the majority of the network have a concensus on its validity. Finally, transactions are immutable, they cannot be revoked and state changes cannot be undone.

In Ethereum blockchain, a transaction allows in addition the creation of a smart contract and stores the results of function calls in smart contracts. Nick Szabo [5] explained that a smart contract is a computerized protocol that executes the terms of a contract. The smart contract enforces the terms of the agreement between untrusted parties about a valid care sale or a loan assessment or voting or health care tracking, etc. Any user can create a smart contract by posting a transaction in the Blockchain (the creation of a smart contract is a type of transaction). Once created, the contract is assigned to a unique address used to interact with that particular contract. An address is a secure identifier, a result of a mathematical operations and the application of encryption algorithms [6]. One could consider a contract as a class in object oriented concepts and each deployment of the contract could be considered as an instance of the object. In other words, a contract may be deployed to a network multiple times, and each instance would have a distinct address. A Smart contract is written in a smart contract-specific programming languages like Solidity for Ethereum. It is then compiled into 'bytecode' that is read and run on the 'ethereum virtual machine' (EVM). A contract can define multiple entry points of execution. In Ethereum's Solidity language, each entry point is defined as a function. A transaction's content will specify the entry point at which the contract's code will be invoked. Therefore, transactions act like function calls in ordinary programming languages. After a contract finishes processing a message it receives, it can pass a return value back to the sender. The contract's state will then be updated accordingly.

Trust in the correct execution of smart contracts extends directly from regular transactions since their creation and execution result from blockchain transactions. Thus, they are immutable. What is more, all their inputs and outputs are through transactions.

C. Blockchain and process choreographies

Business process management (BPM) [8] includes methods, techniques, and tools to support the design, enactment, management, and analysis of operational business processes. BPM is ensured and automated by Business process management systems (BPMSs) especially at the intra-organizational level (enacted by one single organization). Choreographies are a distributed way for the composition and control of business processes in an inter-organizational level where the control is not enacted by one single entity and where many parties that generally do not know neither trust each other conduct each a certain piece of work and maintain an internal state. The global state of the choreography is obtained through the interactions and message exchanges between the different parties involved in the choreography.

In this context, emerging Blockchain technology comes as a promising solution to execute business process choreographies. To begin, the blockchain is a tamper-proof data structure that captures the history and the current state of the network and transactions move the system to a new state. This a very important notion that is essential for business processes. What is more, smart contracts can be used to implement business collaborations in general and inter-organizational business processes in particular. They define the terms of choreography between different entities. Simply put, it describes the roles of all participants and the rules applying to process choreography execution. In addition, the "consensus" applied among nodes in the blockchain has to be reached also in choreographies. Lastly, trust is a major issue when we deal with business process execution. As a positive fact, the mechanisms implemented within the blockchain technology, as demonstrated in previous section II-B, can offer a solution at this level.

III. Related work

Authors in [9] draw a spectrum of different challenges to encounter and opportunities to have in order to execute business processes on the blockchain in a model-driven way along the different phases of the BPM lifecycle: design, implementation,

execution, monitoring, adaptation and evolution. They present as well future research directions to deal with the challenges. In [7], a runtime verification mechanism is developed on top of the Bitcoin blockchain to verify the correct execution of process choreography instances. The approach is augmented with flexibility to make the proposed runtime verification system deal with the dynamic nature of decentralized choreographies and allow runtime adaptation of processes like the selection of partners during process runtime. In contrast, they make some assumptions. A particular participant called the process owner who initiates the business process execution hands over the control to a first suitable partner to have a specific process activity executed. Then, the selected partner pass the execution to another choreography participant to perform the next task and so on.

Smart contracts have been considered as a more suitable solution for the monitoring and execution of choreographies. A smart contract can define the terms of collaboration between different entities i.e., it describes the roles of all participants and the rules applying to process execution. Authors in [12] present an approach to enable the monitoring and execution of collaborative processes specified as choreographies on smart contracts blockchain. The approach is based on a main component named translator taking as input the business process model in BPMN and generates the corresponding "factory" contract according to specific BPMN- Solidity code transformation rules. In fact, as the smart contracts are written to be used to support multiple instances of collaborations where each instance involves a different set of participants, the "factory" paradigm is of a common use when dealing with the smart contracts. The process execution status and exchanges between all involved participants are stored on-chain in the smart contract instance. [13] developed an optimized method of [12] translating a BPMN model to a minimized Petri-net and compiling the latter to a smart contract. The optimizations are done in order to minimize the execution cost (in gas) of smart contracts on the blockchain through minimizing the frequency of data writes on-chain and the number of variables required to capture the process state on-chain.

Some potential benefits of blockchain technology related to process design are for example tools that model business processes in standard BPMN notation and link them to blockchain like Caterpillar [11]. Caterpillar's implementation is based on [12] and [13]. It is an open-source Blockchain-based BPM system that translates the business processes modelled in BPMN into smart contracts written in Solidity language. Caterpillar supports also the execution of collaborative processes of one pool with multiple lanes. However, exchanges between process participants are not performed via message exchanges but through the blockchain. The tool is now available in 3 versions: version 1.0 which is a first demonstrating prototype, version 2.0 [11], a complete implementation of the approach presented in [13] and version 2.1 [14] released lately in which they defined a binding policy of actors to roles to overcome the limitations of the one pool representation. Actors are simply blockchain accounts that are permitted to execute transactions in the blockchain. Roles are the functions performed by the participants in the collaboration. The binding of actors to roles or "nomination" follows a set of rules defined in a textual specification of the binding policy given as input before the deployment of the process model on the blockchain. However, the same presumption as in [7] is made as the "case creator" or the "process owner" as it is called in [7] nominates itself at the moment of the process instance creation and starts nominating a first suitable participant to have some defined task executed.

Regarding the existing works that contain real implementation of complete business process execution engine of collaborative business processes running on top of blockchain we find Caterpillar. However, the latter forces the representation of the input business process to be in imperative language like BPMN. For example, rule-based artifact lifecycle models as demonstrated by [10] can support such interactions in an intuitive and succinct manner. Yet, no implementation of the defended idea exists. Above all, Caterpillar does not support business process choreography notations where the execution of a task requires multiple participants to intervene. Such tasks are represented in the tool as an external subprocess or as it is called in BPMN a call activity. The call activity is then expanded as a normal sequence flow where the actor-to-role binding is applied to each task of the process. Thus, the root process is modeled as a process of processes.

In the following, we will detail the rules that starting from a declarative description such as XML of the logic of our business case, leading to the process choreography model and then the implementation of that logic in Solidity smart contracts.

IV. FROM XML TO SOLIDITY SMART CONTRACTS

In the current section, a descritption of the XML schema of the touristic itineraries is presented in IV-A. The transformation rules are then enumerated in IV-B.

A. Description of the itinerary XML schema

Touristic itineraries to be transformed to Solidity smart contracts are taken from the touristic itinerary planning system CART. All the tours are sequences of attractions in the cities of Loire Valley like Tours, Amboise, etc. They follow the XML schema presented in Fig. 1 and are restricted by user (eventually the tourist) constraints: (i) a period which presents the start and the end of the overall itinerary. (ii) a budget that cannot be exceeded.

The itinerary is viewed as a sequence of steps where each step has a unique identifier and contains or not a Point Of Interest (referred as PoI). A step is also described by two attributes: the start and end time of the step. A PoI has a type (restaurant, hotel, transport or monument like a château), a geographical position (latitude and longitude), prices (fees), opening and closure hours and other related information: a rating, contact and textual description. Some elements are subject to certain conditions like the fees which depend on the age of the user, existence of an handicap situation or the season in which the tourist visits the PoI.

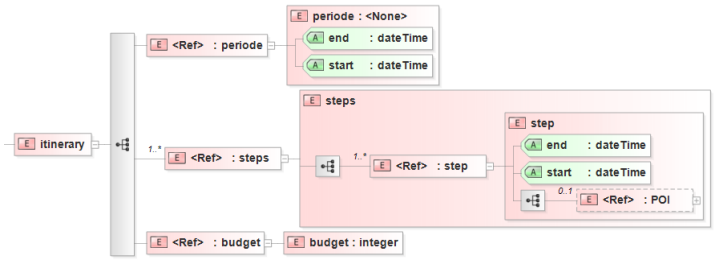

Fig. 1. A touristic itinerary XML schema

To illustrate our case study let's consider the following scenario inspired from [3]: Alice, who lives in Tours, will welcome soon her English friend Gina and she would like to organize a Loire Valley tour to get her friend acquainted with her region and its historic monuments. Alice uses the itinerary recommending system CART[1] to design her personalized tour (see Fig. 2). Then comes the execution of the personalized tour. This is where our work is situated. First, the touristic tour is modeled as a process choreography: Alice will interact with other peers such as the data services instances corresponding to the filtered PoIs given as an output of CART. Alice might also use some public transport application to move between PoIs. The process choreography model elements are then transformed to their respective Solidity code similarly to the rules presented straight after.

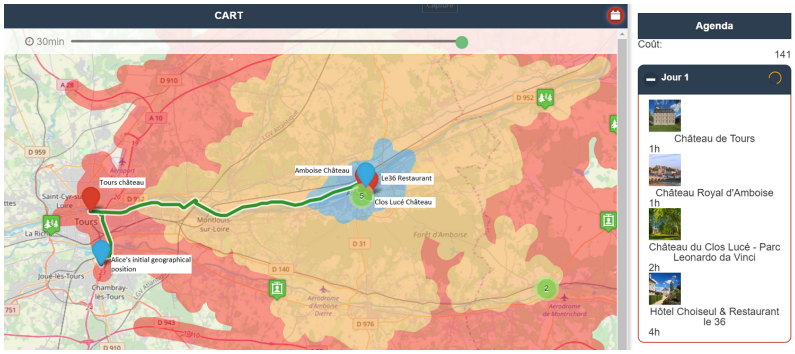

Fig. 2. CART user interface

B. Proposed pattern and transformation rules

Here is presented our main contribution: the transformation rules, first, from XML to process choreography model, then to Solidity code. A summarizing table of the transformation rules is presented in Table. I.

As we mentioned above, the tour is our process choreography model named "Touristic_Visit choreography model" will be transformed to a smart contract baptized "Touristic_Visit smart contract"(rule (a)). The tour is a sequence of steps, where each step containing a PoI is a task in the choreography and will be executed as an external function in the smart contract (rule (b)). When a function is coded as external in Solidity, this means it can be called only externally by other smart contracts. In other terms, when it includes interactions with other contracts. Here interactions are between the main smart contract "Touristic_Visit smart contract" and the "Touristic_Visit oracle contract" that will be detailed just below.

Alice (and her friend) will visit the different PoIs already scheduled. The moving time between the places will be executed internally in the smart contracts (rule (d)). In the available scenarios, no payments are carried out. If it is the case, payments will be represented as choreography task interacting with the payment application (rule (c)).

Other data will be dealt with in the smart contract. Constraints like total visit duration and maximum budget are global variables of the contract (rule(e)).

TABLE I
XML TO SOLIDITY CODE TRANSFORMATION RULES

Rule's identifier	XML element	Choreography element	Solidity code
(a)	Root element : Touristic_Visit	Touristic_Visit choreography model	Touristic_Visit smart contract
(b)	Step containing PoI	choreography task	External function
(c)	Step containing payment	choreography task	External function
(d)	Step with no PoI	choreography task	Internal function
(e)	User's constraints (budget, total visit duration, etc)	embedded in model documentation	Global variable
(f)	PoI attributes (string, numeric, etc)	data perspective	Result of the callback performed by Touristic_Visit **oracle** contract

Tasks standing for steps with PoIs in the process choreography model interact with external resources. Thus, regarding the data perspective we find:

1) data received from external user (eventually the tourist) and to be passed to the smart contract, e.g. Alice will give information about her age, special situation, presence in a group, etc., that will be needed to proceed with the execution of the process choreography model.
2) data received from external applications and services to pass similarly to the smart contract, e.g., data related to a payment task or transport modes.
3) data read from the smart contract. This data is already stored in the blockchain.

To deal with external data, special contracts called **oracles** will be used. Oracles are real-time data feeds that act as mediator between smart contracts and external world due to the fact that smart contracts cannot call external programs directly. Oracles are designed to be used in smart contracts. References [12], [13] called these special programs triggers regarding the fact that they provide external data and trigger smart contract executions. In more details, oracles forward a request from external application to the smart contract via a Solidity event and receive corresponding response via a contract function call. Consequently for each generated choreography_smart contract, an oracle is also generated to deal with external data. Attributes linked to a PoI like fees, duration of the visit and opening hours will be part of the result of the callback

[1]CART:https://smartloire.firebaseapp.com/cart.html

performed by the oracle contract named “Touristic_Visit oracle contract”(rule (f)).

V. Running example and implementation details

The XML file corresponding to the touristic itinerary generated by CART in Fig. 2 is our starting point. Transformation rules {XML $\Longrightarrow$ choreography} in Table. I are first applied to produce the process choreography model in Fig. 3 where we can see that Alice decides to explore first the oldest district of Tours by visiting the château of Tours. Then, she chooses the east of Tours to visit châteaux of Amboise and Clo-Lucé in Amboise city. At the end of the day, Alice prefers to go to a local restaurant, “Le36 restaurant” in “Le Choiseul hotel”. Alice chooses to use public transport to move from the château of Tours to the château of Amboise. For the rest of the places of interest, Alice will not use any mean of transport given the fact that the places are in the same zone (see Fig. 2).

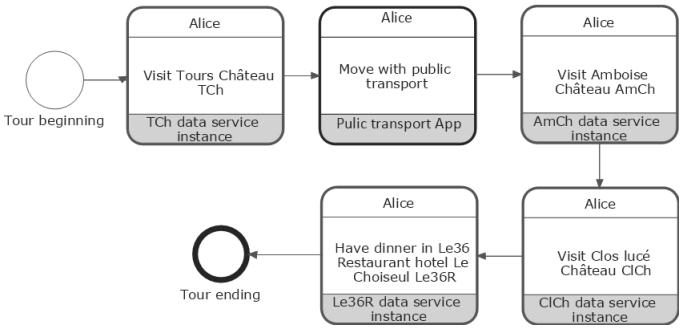

Fig. 3. Process choreography model of a touristic visit

The process choreography model of a touristic visit is a sequence of tasks where participants are the tourist (in our case Alice), PoIs’ data services instances and public transport application. A choreography task is either the visit to a PoI or a moving task.

The second part of the transformation rules {choreography $\Longrightarrow$ Solidity smart contract} in Table. I are solicited to move from process choreography model to smart contract in Solidity as explained below.

For demonstration purpose, we deployed and implemented our “TouristicVisit_contract” in Caterpillar. We used the last version of the tool [14] that requests the application of an actor-to-role binding policy along with the process instance. Actors in our case are the end-user and the service providers of the different services. Roles are in number of five: tourist, and four other roles relative to the four PoIs data services instances.

An extract of the generated smart contract in given in Listing. 1. Our smart contract named “TouristicVisit_contract” contains, according to the transformation rules, four external Solidity functions to start the execution of the tasks containing PoI and one internal function for the moving task. Line 35 in Listing. 1 stands for the execution of the first task “Visit Tours château TCh”. Here the moving task is considered as a task with no PoI. But for a more realistic implementation, it should be implemented as a task interacting with external application.

```
1  contract TouristicVisit_Oracle_Abstract_contract{
2      function Visit_TCh_query_service(uint, uint, bytes32
           ) external returns (uint));
3  ...}
4
5  contract TouristicVisit_contract{
6      //gloabal variables
7      uint public marking = uint(4);
8      uint public startedActivities = 0;
9      address internal oracleAddress;
10     mapping(uint => address) oracleAddresses;
11     uint cumulatedFee = 0;
12     uint cumulatedVisitDuration = 0;
13     uint age = 0;
14     bytes32 particularSituation;
15     ....
16     function TouristicVisit_contract(){
17         //TouristicVisit_Oracle_contract instance
               address
18         oracleAddresses[2] = 0
               x98c9c18f0e3e7c8deec7b2dc9ac5500e2f1fb62c;
19         oracleAddress = oracleAddresses[2];
20     ....}....
21     function Visit_TCh_callback(uint serviceIndex, uint
           _fee, uint _visitDuration, uint _openingHour,
           uint _closureHour) external returns (bool){
22         var(tmpMarking, tmpStartedActivities)= (marking,
               startedActivities);
23         if (serviceIndex == uint(1)){
24             //TouristicVisit_Oracle_contract instance
                   performs the callback to the
                   corresponding service
25             require(msg.sender == oracleAddress &&
                   tmpStartedActivities & uint(2) != 0);
26             {//code to execute like updating
                   cumulatedFee and cumulatedVisitDuration
27             }
28             step(tmpMarking | uint(8),
                   tmpStartedActivities & uint(~2));
29         ...}} ....
30     function step(uint tmpMarking, uint
           tmpStartedActivities) internal{
31         while(true){
32             if (tmpMarking & uint(4) != 0){
33                 uint reqId=Visit_TCh_query_service(1,
                       age, particularSituation);
34                 tmpMarking &= uint(4);
35                 tmpStartedActivities |= uint(2);
36             }.....}}
37 }
```

Listing 1. TouristicVisit contract

Another special smart contract named “TouristicVisit_Oracle” is implemented to handle the data exchanges and interactions between the “TouristicVisit_contract” and the off-chain world, here, the data services instances of the PoIs. We applied the rule related to the external data perspective (rule (f) in Table. I) and used the recommendations proposed in [11]. The TouristicVisit_Oracle contract is completely coded and deployed to the blockchain before the main contract i.e., TouristicVisit_contract. Then the address of the oracle along with its Application Binary Interface (ABI) should be provided to the main contract. The ABI is a JSON file that describes the functions of a deployed contract (their inputs, outputs, state mutability: read-only or write access, etc.). It allows to contextualize the contract and call its functions. Listing. 2 shows an extract of the oracle contract’s code related to the first task “Visit Tours château TCh” (line 9 in Listing. 2). The data to bring to the blockchain via the callback are information like fees, duration of the visit and opening hours (line 22 in Listing. 1).

The “Oracle_Abstract_contract” presented in top of the

"TouristicVisit_contract" is an abstract contract containing specially headers of functions in the oracles contracts. In Solidity, abstract contract are close-similar to abstract classes in object-oriented programming language.

Global variables (lines 7 to 14 in Listing. 1) and data related to the process state are stored on-chain. The user could proceed with the execution of the process at any time by refreshing its instance. This could be very advantageous in the case of itineraries lasting more than one day. Scenarios of two days or more are also implemented but not presented in this paper.

```
1  contract TouristicVisit_Oracle{
2      struct Request{
3          uint serviceIndex;
4          address oracleInstanceAddress;
5      }
6      Request [] requests;
7      event Visit_TCh_Requested(uint, uint,bytes32);
8      ...
9      function Visit_TCh_query_service(uint serviceId,
           uint age, bytes32 particularSituation) external
            returns (uint){
10         uint index = requests[serviceId].serviceIndex;
11         msg.sender = requests[serviceId].
               oracleInstanceAddress;
12         requests.push(Request(index, msg.sender));
13         ....
14     }.....
15 }
```

Listing 2. TouristicVisit oracle contract

VI. Discussion and conclusion

This research work is aimed at proposing a pattern or a set of transformation rules to transform touristic itinerary generated in declarative language such as raw XML to blockchain smart contract in Solidity programming language.

Our choice of blockchain technology is due to its opportunities for BPM in general and process choreographies in particular. First, as a tamper-proof data structure, blockchain captures the history and the current state of the network and transactions move the system to a new state. This could be used to execute touristic itineraries of more than one day. The execution state of the process instance is maintained on-chain and the process execution is carried on at any time just by refreshing the instance. Additionally, the consensus applied among nodes in the blockchain which is the same as negotiation between collaborating partners has to be reached also in choreographies. Then, smart contracts technology was used to implement the business logic of our touristic itineraries. Smart contracts, as they are designed, can define and describe the terms of a choreography between different entities by describing the roles of all participants and the rules applying to process choreography execution. Moreover, the blockchain is considered trustworthy not only as an infrastructure but also as a coordination mechanism. This is especially due to the consensus mechanism and the immutable nature of transactions, including amongst others smart contracts as all their inputs and outputs are through transactions.

At the same time, blockchain technologies are under active development and many challenges are revealed regarding its use for the deployment and execution of process choreography. Primary, blockchain transactions are not cost-free. This is a very import issue to consider in our case regarding the fact that the execution of touristic itineraries may include payments and use of transport applications. Although the latter are external to the blockchain, interactions between them and the process choreography are on-chain as they extend from blockchain transactions. Trust insurance in the blockchain has also been criticized. Trust is essential in our case. Process choreographies are of open and untrusted nature where trust relations change dynamically.

To conclude, we are currently working on the development of a complete template to transform touristic itinerary to Solidity smart contract language. More diversified choreography tasks like moving between PoIs using an urban mobility application or carrying out payments are in the scope of future work. We will intend in the same way to examine the (scalable) performance cost due to the execution of smart contracts on the blockchain. Semantic verification of the transformation outputs (first process choreography models and then smart contracts) are two other work perspectives.

Acknowledgment

The present work is developed in the context of the Smart Loire project [2].

References

[1] Boulakbech, M., Messai, N., Sam, Y., Devogele, T., & Etienne, L. (2016, June). SmartLoire: A Web Mashup Based Tool for Personalized Touristic Plans Construction. WETICE 2016 (pp. 259-260). IEEE.

[2] Smart Loire project, https://intelligencedespatrimoines.fr/smart-loire-apr-ir-2017/ - last accessed April 2018.

[3] Boulakbech, M., Messai, N., Sam, Y., & Devogele, T. (2017, June). Visual configuration for restful mobile web mashups. ICWS 2017 (pp. 870-873). IEEE.

[4] Yermack, D. (2017). Corporate governance and blockchains. Review of finance, 21(1), 7-31.

[5] Szabo, N. (1994). Smart contracts. Unpublished.

[6] Dannen, C. (2017). Introducing Ethereum and Solidity. Berkeley: Apress.

[7] Prybila, C., Schulte, S., Hochreiner, C., & Weber, I. (2017). Runtime verification for business processes utilizing the bitcoin blockchain. Future Generation Computer Systems.

[8] Van Der Aalst, W. M., Ter Hofstede, A. H., & Weske, M. (2003, June). Business process management: A survey. BPM 2003 (pp. 1-12). Springer, Berlin, Heidelberg.

[9] Mendling, J., Weber, I., Aalst, W. V. D., Brocke, J. V., Cabanillas, C., Daniel, F., ... & Gal, A. (2018). Blockchains for business process management-challenges and opportunities. ACM (TMIS) 2018, 9(1), 4.

[10] Hull, R., Batra, V. S., Chen, Y. M., Deutsch, A., Heath III, F. F. T., & Vianu, V. (2016, October). Towards a shared ledger business collaboration language based on data-aware processes. ICSOC 2016 (pp. 18-36). Springer, Cham.

[11] López-Pintado, O., Garca-Bauelos, L., Dumas, M., Weber, I., & Ponomarev, A. (2018). CATERPILLAR: A Business Process Execution Engine on the Ethereum Blockchain. arXiv preprint arXiv:1808.03517.

[12] Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., & Mendling, J. (2016, September). Untrusted business process monitoring and execution using blockchain. BPM 2016 (pp. 329-347). Springer, Cham.

[13] Garca-Bauelos, L., Ponomarev, A., Dumas, M., & Weber, I. (2017, September). Optimized execution of business processes on blockchain. BPM 2017 (pp. 130-146). Springer, Cham.

[14] López-Pintado, O. (2018). Caterpillar source code (Version 2.1) [Source code]. https://github.com/orlenyslp/Caterpillar/tree/master/v2.1- last accessed 30 December 2018.

[15] Tran, A., Lu, Q., & Weber, I. (2018). Lorikeet: A model-driven engineering tool for blockchain-based business process execution and asset management. BPM Demo Track.