Smart Auctions for Autonomic Ambient Intelligence Systems

Article

Accepted version

A. Bordonaro, A. De Paola, G. Lo Re, M. Morana

In Proceedings of IEEE International Conference on Smart Computing (SMARTCOMP), Bologna, Emilia-Romagna, Italy, 2020, pp. 180-187

Smart Auctions for Autonomic Ambient Intelligence Systems

Antonio Bordonaro, Alessandra De Paola, Giuseppe Lo Re, Marco Morana
Department of Engineering, University of Palermo, Italy
Email: antonio.bordonaro@unipa.it, alessandra.depaola@unipa.it, giuseppe.lore@unipa.it, marco.morana@unipa.it

***Abstract*—The main goal of Ambient Intelligence (AmI) is to support users in their daily activities by satisfying and anticipating their needs. To achieve such goal, AmI systems rely on physical infrastructures made of heterogenous sensing devices which interact in order to exchange information and perform monitoring tasks. In such a scenario, a full achievement of AmI vision would also require the capability of the system to autonomously check the status of the infrastructure and supervise its maintenance. To this aim, in this paper, we extend some previous works in order to allow the self-management of AmI devices enabling them to directly interact with maintenance service providers. In particular, the combination of smart contracts and blockchains enables AmI systems to autonomously communicate with untrusted entities and complete secure transactions without the brokering of a trusted third party. The proposed approach has been validated in the sample case of an AmI application responsible for managing requests from faulty devices in a Smart home.**

I. Introduction

One of the main requirements of Ambient Intelligence (AmI) [1] is to support users in their daily activities, while guaranteeing a low level of intrusiveness, and minimizing the need for manual intervention in management tasks [2], [3]. The achievement of these goals would requires the AmI system to include also sophisticated mechanisms for self diagnosis and maintenance.

The sensing and actuating devices of an AmI systems, as well as the appliances of a smart home, for instance, may exhibit some faults that (i) need to be detected and (ii) fixed by a technical intervention. Whereas the automatic management of such a situation can be easily performed inside a trusted realm, transactions that involve external entities pose several challenges because of the presence of untrusted entities.

In particular, current AmI systems are not able to automatically negotiate with untrusted third parties the execution of specific tasks, such as service providing or money transfer [4]–[6].

In this paper, we address such a challenge by creating and managing *smart auctions* through smart contracts and blockchain technology. The solution we propose allows the AmI system both to place public requests for devices maintenance and select the most convenient offer, according to a certain evaluation policy.

A Smart Contract (SC) [7], [8] is a program deployed over a distributed ledger, that enables unknown parties to automatically agree on the transfer of digital assets under given conditions. In the scenario addressed here, the adoption of SCs guarantees that the agreement between service provider and consumer is honored. Several blockchain technologies have been proposed in order to enable the creation of a secure distributed ledger and to support the creation of SCs [9], [10]. Among them, Ethereum is one of the most used, due to the availability of a Turing-complete language which allows for the creation of complex customized contracts [11]. Furthermore, such a specific feature of Ethereum enables the development of SCs whose automatic execution is conditioned by the fulfillment of some trigger conditions.

Smart auctions implemented through Ethereum SCs allow the participation of unknown and untrusted entities, while guaranteeing a high level of security and also binding the parties to the agreed terms. Moreover, differently from other distributed paradigms, such as Service Oriented Architectures, this solution does not require the involvement of a third-party centralized and trusted entity. For this reason, SCs have been successfully adopted in many fields, such as legal processes, crowdfunding agreements and financial derivatives [12].

According to the proposed approach, the AmI system is able to react to events reported by the physical infrastructure, and to interact with external service providers responsible for its maintenance. More specifically, when a faulty device is detected, a self-reasoning mechanism is started in order to evaluate the severity of the fault and its consequences for the whole system functioning. Then, through a probabilistic approach, the AmI system plans the best recovery strategy; if an external maintenance intervention is required, a *smart auction* is started by creating a SC aimed to select the best bid among different providers.

The remainder of this paper is organized as follows. Section II reviews the literature about process automation in smart homes, and the adoption of blockchains and smart contracts in other application fields. Section III provides a brief description of the blockchain technology, and shows how smart contracts allow to enable automated and secure interactions between autonomous systems. Section IV describes the architecture of the proposed system and its main features. Section V details the features of the proposed smart auction. Finally, Section VI draws our conclusions and proposes directions for future work.

II. Related work

A blockchain is a distributed data structure shared among entities/nodes of a distributed system, which guarantees the integrity of stored data. Originally introduced to solve the *double-spending* problem for the Bitcoin cryptocurrency [10], blockchains enable decentralized and secure management of transactions [13].

In traditional distributed systems, this task requires the brokering of a trusted entity, whose presence implies some transaction fees and poses a variety of security issues due to the presence of a single point of failure. On the contrary, secure transactions validation between untrusted participants is achieved by blockchains without the involvement of a third trusted entity. Moreover, blockchains ensure the *pseudonymity* of users, by identifying them by an *address* that has no relationship with their real identity, and guarantee the *persistency* of the data, i.e., when a transaction is successfully validated and added to the blockchain, it becomes immutable and can not be modified or removed.

Different types of blockchain have been proposed and developed and today they can be classified into *public*, *private*, and *consortium* blockchains. The main difference between them is the level of access granted to participants. *Public* blockchains (e.g. Bitcoin, Ethereum) are fully open and anyone is free to join them, execute new transactions or validate existing ones. In this scenario, users are incentivized to freely contribute to the maintenance of the infrastructure through reward mechanisms. A public blockchain is also defined *permissionless*, since every node can perform transactions, read the blockchain, and participate to the consensus algorithm without performing any preliminary authentication phase.

On the contrary, private and consortium blockchains are defined *permissioned* since only a group of selected nodes can contribute to the consensus algorithm. In particular, private blockchains allow users belonging to the same organization to access the blockchain through an authenticated and verified invitation. In this case, the organization that maintains the network infrastructure has complete control of the blockchain and can arbitrarily add, modify or delete data. Consortium blockchains have a similar setting, but they involve a consortium of different organizations; thus, the blockchain can be considered partially distributed among them. Access to permissioned blockchains is allowed after an authentication phase; each user is enabled to perform only specific activities and the infrastructure owner has complete control over the operations performed by each user. In this case, pseudonymity of users and decentralization are not guaranteed and a trusted entity, or a consortium of trusted entities, is required.

Permissioned blockchains can process much higher transactions per second (TPS) than public ones, since the small number of authorized entities results in significantly lower time to acquire distributed consensus. Nevertheless, they do not exhibit the main advantages of their public alternative, i.e., a decentralized architecture and the pseudonymity of users. Thus, they can be just seen as secure distributed databases which rely on a centralized trusted authority. As a consequence, public blockchains are usually preferable and are particularly suitable for many application scenarios, such as Ambient Intelligence.

Besides allowing decentralized transactions of cryptocurrencies, blockchains are the enabling technology to support the creation of smart contracts, i.e., distributed software that is automatically executed under certain conditions. Smart contracts were firstly defined in [8] as machine-readable transaction protocols which create a contract with predetermined terms. This technology allows two untrusted parties to make an agreement without the need for a trusted third party. Moreover, once the contract has been validated, it can not be modified or removed, that is it can not be retreated.

Blockchain technology and smart contracts have been recently applied to many fields, such as healthcare, finance, and e-government, in order to provide interaction security, user anonymity and data integrity, through a fully distributed approach. The authors of [14], for instance propose the adoption of smart contracts to develop a fully decentralized electronic voting system. Authors state that their system allows to maximize users privacy, also avoiding the necessity of a trusted authority that coordinates the voting process and computes election results. The authors of [15] propose a distributed system based on the Ethereum blockchain which aims to validate experimental evaluations of new healthcare solutions, by guaranteeing soundness and integrity of obtained results. In [16], a business model for trading electricity through smart contracts is proposed. Such a model is particularly relevant since it allows the distribution of resources in a competitive domain in which sales price transparency and the need for trust between prosumers are required.

Conversely, blockchains and smart contracts have not been yet fully exploited in Internet of Things and Ambient Intelligence scenarios. Only few works, indeed, propose the adoption of such technologies to guarantee secure interactions among IoT devices and AmI systems [17], [18]. Recently, the authors of [19] proposed a completely decentralized system for secure communication among IoT devices, by defining secure virtual zones (*bubbles of trust*). Communications are considered secure only if they occur between devices in the same zone. Thus, a bubble of trust is a group of devices that can trust each other. Bubbles of trust are obtained by exploiting smart contracts in order to write secure code to be executed by IoT devices, while blockchain transactions allow to perform secure exchanges of information.

III. Blockchains and smart contracts

From a technical point of view, a blockchain can be seen as a sequence of linked blocks, each containing a *content*, i.e., a list of transactions and some metadata, and a *reference* to the predecessor, as identified by its hash value. Each node in the blockchain network is characterized by a pair of public/private keys that are required to sign each transaction started by the key owner and verify the authenticity of the signed transactions respectively. The validation process, which allows to add new

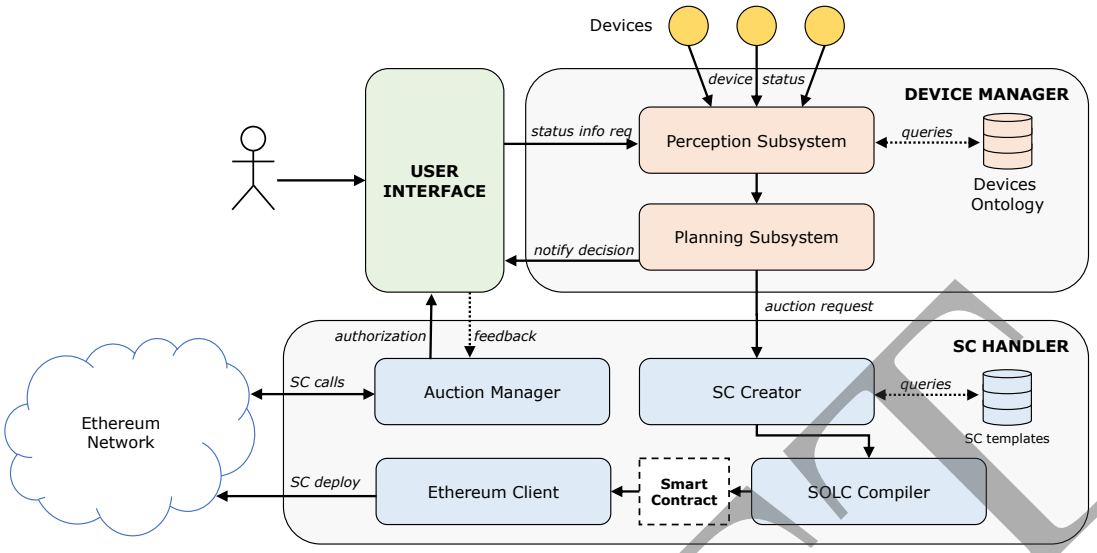

Figure 1: System Architecture. The Device Manager provides the smart home with the core AmI functionalities; the SmartContract (SC) Handler is responsible for evaluating the requests coming from the planner and managing all the phases of the smart auctions.

blocks to the blockchain, is performed by special nodes, named *miners*, after solving a specific mathematical problem. In the case of bitcoin, for instance, such a challenge is named *proof of work* (PoW) and is a way to implement distributed consensus.

Several blockchain technologies have been proposed to support smart contracts development. Among them, Ethereum [9] is an account-based blockchain platform that is widely used since it enables the creation of complex smart contracts.

Ethereum smart contracts are not considered as something to be fulfilled but rather as autonomous entities, always in execution, living in the Ethereum network. When a contract account receives a message, its code is activated, allowing it to access its internal memory, to send other messages, to perform transactions, or to create new contracts.

The contract code in Ethereum is written in a low-level, stack-based bytecode language, referred to as *Ethereum virtual machine code* or *EVM code*. The most common approach to write smart contracts is to adopt an high-level language (e.g. Solidity) and then compile smart contracts into EVM code. The deployment of a smart contract is performed through a transaction in which the *data* field contains EVM code of the smart contract. After the deployment, the smart contract is univocally identified in the Ethereum network by its address, and other network nodes can interact with it by calling its methods, which are available through its ABI (Application Binary Interface).

IV. System Architecture

In AmI systems, many networked devices, pervasively deployed in the environment, interact in order to gather information and perform complex tasks [20], [21]. IP cameras, indoor and outdoor environmental sensors, thermostats, HVAC systems, lighting systems, household appliances, electrical sockets, and intelligent security systems are just a few examples of devices used in such contexts. Data coming from the physical infrastructure are exploited by AmI systems to define actions aimed to satisfy and anticipate the users' needs. In order to achieve such goal, however, it is necessary that AmI systems be able to autonomously check the status of the devices and supervise their maintenance. Taking inspiration from the Autonomic Computing paradigm [22], we propose a novel approach to allow an AmI system to self-manage its own physical infrastructure by means of ad-hoc smart auctions.

The core components of our architecture are shown in Fig. 1. The user can interact with the AmI system through a *user interface* which allows to perform some common tasks, such as monitoring the status of the devices or sending commands to them. Devices are controlled by the *device manager*, which includes the AI algorithms at the core of the AmI system and consists of three different components, namely the *ontology* of the devices, the *perception* and the *planning* subsystems.

The *ontology* allows the AmI system to own an explicit model of itself, of the surrounding environment, and of the different ways it can interact with users. In particular, ontologies make it easy to represent such knowledge in an efficient and machine-computable way, by formally defining the relationships among set of terms belonging to a specific domain. The ontology we adopted provides a representation of the structural organization of devices, as extensively described in [23], [24]. It also describes how data flows within the system, highlighting the relationships between devices and the monitored (or controlled) environmental properties. Furthermore, it models the *task* concept, i.e., each action that the AmI system can execute.

Tasks are implicitly associated with the devices involved

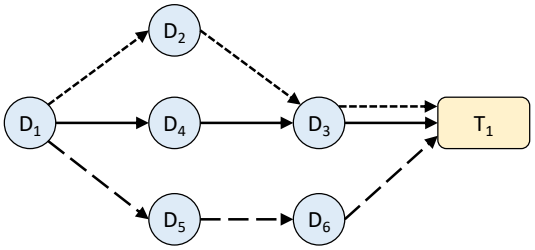

Figure 2: Example of device dependencies graph for a given task.

in their execution; thus, *dependency graphs* such the one shown in Fig. 2 can be built. Each task t_j can be also characterized by a relevance value $w_t(T_j) \in [0,1]$, that weights the importance of the task as rated by the administrator of the AmI system. Higher scores, for instance, are assigned to tasks that implement critical functionality (e.g. surveillance system), while lower values are given to secondary functions (e.g. lighting management).

Information modeled by the ontology is exploited by the *perception subsystem*, whose aim is to process data provided by physical devices and represent them at a higher level of abstraction in order to describe both the status of the environment and of the system itself. The *perception subsystem* plays a crucial role in implementing the *monitor-analyze-plan-execute* cycle at the basis of the autonomic behavior of our system. It adopts a rule-based approach to analyze the state of the physical infrastructure according to a parameter set, including, for instance, the degree of accuracy of the monitored information, the state of the devices, the relevance and redundancy of devices, their energy consumption, and the residual lifetime of the battery-powered devices.

Each device D_i involved in the execution of a task T_j is associated with an *impact* value $I(D_i, T_j) \in [0,1]$, that represents the contribution of D_i to the achievement of T_j. This value strictly depends on the redundancy of the *dependency graph* of the task T_j. In particular, if N_j is the number of parallel paths in the *dependency graph* of T_j, and N_j^i is the number of these paths that includes D_i, the *impact* value can be calculated as follows:

$$I(D_i, T_j) = \frac{N_j^i}{N_j}. \quad (1)$$

For example, the *dependency graph* in Fig. 2 shows the existence of three distinct paths to achieve the task T_1, then we can compute $N_1 = 3$. The device D_2 is part of one of these paths, and its *impact* value is $I(D_2, T_1) = 1/3$. That means that the fault of D_2 is not critical, since there are two other ways to complete the task. On the contrary, the device D_1 is highly critical, since it is involved in all the possible paths to achieve T_1; consequently, D_1 is marked with the maximum *impact* value, i.e., $I(D_1, T_1) = 1$.

According to the relevance $w_t(T_j)$ of each task j, and to the impact $I(D_i, T_j)$ of a given device i on that task, the *perception subsystem* evaluates the overall *relevance* of each device to the functioning of the whole AmI system as:

$$w_d(D_i) = \max_{T_j} \{I(T_i, T_j) \cdot w_t(T_j)\}. \quad (2)$$

Such a knowledge is used in a *self-reasoning* mechanism in order to detect critical faults. To this aim, we adopted a rule-based inference engine based on Jess (Java Expert System Shell) [25], which allows to express logical rules with a LISP-like syntax, and uses a pattern-matching algorithm to query the knowledge base. Each fact of the knowledge base is a true proposition about the state of the system, and according to the following specific templates:

- *static knowledge templates*, used for information gathered during the setup of the system and expressed through the ontology, such as the relevance of the tasks, the impact value of each device on each task, and the overall relevance of each device;
- *dynamic knowledge templates*, used for information that is continuously updated at runtime, e.g., that received from the devices;
- *alert templates*, used for information sent to the *planning subsystem* in order to trigger the planning of a possible maintenance intervention.

Rules used by the *perception subsystem* follow a "*if <conditions> then <action>*" form; thus, a rule is activated only if all its conditions are satisfied. Rules are evaluated once for a given set of facts, and their evaluation is repeated only after the addition of new facts to the knowledge base. Through these rules, static and dynamic knowledge are exploited by the *perception subsystem* to infer the state of physical devices. This is represented through a property named *device-condition*, whose values correspond to specific levels of alert. A *stress* condition, for instance, causes a critical alert that must be promptly addressed by the *planner subsystem*, an *attentive* condition corresponds to a medium alert that can be fixed by a non-urgent intervention, while a *normal* condition represents the best scenario in which no intervention is required. The adopted rules trigger the transition between the possible states of the *device-condition* property. It is worth noting that different finite-state automata model the evolution of the *device-condition* property for different classes of devices.

Battery-powered devices, for instance, have to be monitored with respect to their residual amount of energy. Then, a *stress* condition could occur when a low battery value is registered for a device with a *high* relevance, i.e., a device that is crucial to at least one critical task. Conversely, if the relevance of the device is low, its battery depletion would just lead to an *attentive* condition. In order to trigger these alerts, the following rules are adopted:

defrule setStressCondition-BatteryPoweredDevices:
if (*energy-level*(D_i) is "low") **and** ($w_d(D_i)$ is "high") **then**
device-condition(D_i) ← "stress"

defrule setAttentiveCondition-BatteryPoweredDevices:
if (*energy-level*(D_i) is "low") **and** ($w_d(D_i)$ is "low") **then**

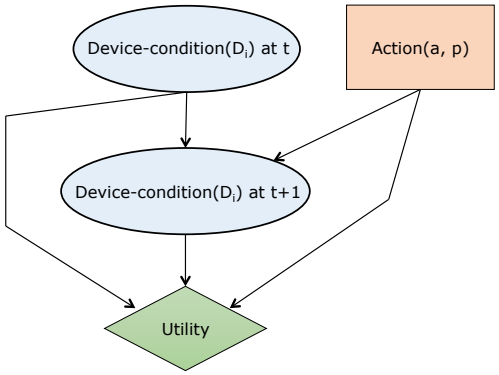

Figure 3: The *planning subsystem* influence diagram.

device-condition(D_i) ← "attentive"

Besides conditions related to the energy level, the *perception system* has to manage alarms triggered by other device-specific faults. To this aim, the *ontology* contains the definition of all the alarm signals a device could send, each associated with a criticality degree provided by the producer. This type of alarms are triggered through rules like the following:

defrule setStressCondition-PoweredDevices:
if ($\exists$ fault signal f_i triggered by device D_i) **and**
(*criticalness*(f_i, D_i) is "high') **and** ($w_d(D_i)$ is "high") **then**
device-condition(D_i) ← "stress"

defrule setAttentiveCondition-PoweredDevices:
if ($\exists$ fault signal f_i triggered by device D_i) **and**
(*criticalness*(f_i, D_i) is "low') **and** ($w_d(D_i)$ is "high") **then**
device-condition(D_i) ← "attentive"

The knowledge inferred by the *perception subsystem* is then managed by the *planning subsystem*, which is responsible for selecting the appropriate actions to be taken in order to bring the system back to a *normal* condition. The planner evaluates if a triggered alarm can be tackled through a simple human intervention or if it requires an external maintenance service. In the former case, the *planning subsystem* notifies the AmI system's administrator providing the details of the request and suggestions on how to deal with it. Otherwise, the planner forwards the request to the *smart contract handler*, which will instantiate an appropriate smart contract in order to request third-party assistance.

The selection of the action to be taken mainly depends on two factors: guidelines provided by the producer of the device, and history of past interventions. Since the effectiveness of a specific action, given a device condition, is characterized by a non-negligible level of uncertainty, we chose to design the planner inference core according to a probabilistic approach. In particular, the *planning subsystem* is based on an influence diagram [26], a generalization of a Bayesian network, capable of supporting probabilistic decision-making.

According to the influence diagram structure shown in Fig. 3, the probability of a change of the *device-condition* property for a device D_i, given the current state and the possible action a performed by the potential external/internal provider p, is given by:

$$p(\textit{device-condition}(D_i, t+1) \mid \textit{device-condition}(D_i, t), \textit{action}(a, p)). \quad (3)$$

The conditional probability distribution of the state transition model is initialized by considering the suggestions of the device producer, and is updated after each maintenance intervention. This continuous training process allows to include in our planning model the experience derived from past interaction with service providers, thus implicitly rewarding reliables providers associated with a high rate of successful interventions. The utility node gives a score to each state transition, considering not only such past experience, but also the overall reputation of potential external providers, gathered through a distributed reputation management system that involve a collaborative network of AmI systems [27], [28].

By exploiting the state transition model and the utility function, the influence diagram allows the *planning subsystem* to select the best action to perform in order to maximize the expected utility value.

When the selected action requires the intervention of an external provider, the *planning subsystem* triggers the *smart contract handler* sending it a list of potential providers, obtained by considering only those with a reputation higher than a threshold chosen by the administrator of the AmI system.

The *smart contract handler* is the only system component that can directly interact with the Ethereum network; it compiles the smart contracts through *SOLC* (Solidity command-line compiler), producing a bytecode executable by the Ethereum Virtual Machine, and the ABI (Application Binary Interface) public interface. Finally, the smart contract is deployed in the Ethereum network by means of the *GETH* client [29], developed by Ethereum Foundation developers in the Go programming language, which also provides support for a full integration in mobile applications. After the deployment, the system publishes the address and the ABI of the smart contract in a public repository, known to service providers, so that they can participate in the smart auction.

V. Smart Auctions

As mentioned in the previous sections, the maintenance interventions are managed by the autonomic AmI system through smart auctions; such an approach guarantees that transactions are executed in a secure way and also that the agreement negotiated between the two parties will be honored.

The procedure followed to manage a smart auction is summarized in Fig. 4. When a fault occurs, the involved device sends a signal to the smart home; here, the *perception subsystem* analyzes the severity of the current state and determines whether to trigger a smart auction through the *planning subsystem*. Each auction remains valid for a given period, during which service providers can submit their bids. When the auction ends, the smart contract automatically selects the best bid and informs the AmI system, which in turn requires the authorization of the AmI system administrator. Once the

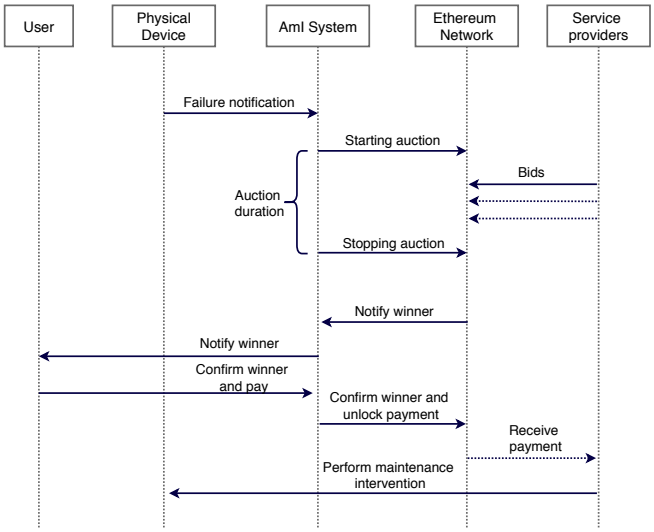

Figure 4: Flow diagram of the smart auction.

provider has been confirmed, the smart contract code notifies the winner and unlocks the conditional payment. Only after the maintenance intervention has been completed, the payment is done.

Auctions are managed through a set of smart contract templates that can be customized to deal with different kind of events.

Service providers can submit an offer by invoking the *bid()* function. As shown in Fig. 5, bids can be submitted only during the auction life-time (line 3), and only by authorized service providers (line 6), that are identified by the *planning subsystem* according to their distributed reputation. Moreover, the smart auction guarantees that providers can submit a single bid per auction (line 9). Implicitly, due to the lack of functions that enable providers to modify or read the bid list, they can not retract bids already placed, and are not able to know the bids submitted by others.

Each Ethereum transaction performed to submit a bid involves a small fee (less than $0.10 in average during the last six months), that is paid by the service provider. Nevertheless, this cost can be recovered as commission for the service intervention and it is necessary to discourage fraudulent service providers from participating in the auction.

When the auction ends, the SC handler declares the conclusion by invoking the smart contract *auctionEnd()* function (see Fig. 6). This function can be called only by the SC handler, which is the owner of the SC (line 5), and is aimed to trigger the identification of the best bid by means of the *winnerIdentification()* function (line 17).

Identifying the winner (see Fig. 7) is a critical task that must consider several issues in order to guarantee the resistance of the smart auction to security attacks performed by malicious providers. The first, for instance, is that the smart auction has to include also a minimum threshold for bids, below which they are ignored (line 12). This constraint, tunable by the AmI

```
1  function bid(uint amount) public {
2      //  Is the auction is done?
3      require(!ended);
4
5      // Is the user blacklisted?
6      require(authorizedProvider(msg.sender));
7
8      // Has the user previously placed a bid
9      require(!hasBid(msg.sender));
10
11     // Update bidders list
12     bidderUsers[bidderUsersCount] = msg.sender;
13     bidderUsersCount = bidderUsersCount + 1;
14
15     // Store the bid
16     bids[msg.sender] = amount;
17 }
```

Figure 5: SC function to place a bid for the smart auction.

```
1  function auctionEnd() public {
2
3      /*  Only the beneficiary of the repair
4          service can interrupt the auction */
5      require(msg.sender == owner);
6
7      //  Auction not yet ended
8      require(now >= auctionExpirationTime);
9
10     // auctionEnd has already been called
11     require(!ended);
12
13     // Set variabile flag a true
14     ended = true;
15
16     // Establish the best bid
17     winnerIdentification();
18 }
```

Figure 6: SC function to end the auction and notify the owner of the best bid.

system administrator, allows both to discard zero bids which would invalidate the auction, and to neglect bids for which the quality of service may be not adequate.

In the implementation discussed here, the best bid is that associated with the lowest price (lines 11-22), and it is considered valid only if it does not exceed the maximum threshold set by the AmI system administrator (line 25). It is worth noting that such policy may be simply modified by adopting different criteria, for example by performing a multi-objective optimization that takes into account also the provider reputation.

Finally, if the selected proposal is accepted by the user, the payment process is activated by transferring the required amount of ETHs to the smart contract, which has to be at least equal to the maximum threshold applied to bids. However, the reward remains bound into the smart contract account until the conclusion of the maintenance intervention. Then the AmI system interacts with the smart contract notifying the correct conclusion of the requested service by means of the *unlockPayment()* function, as shown in Fig. 8. Even this

```
1  function winnerIdentification() private returns (
       bool) {
2    bool foundBid = false;
3
4    // Check if there's at least one bid
5    if(bidderUsersCount == 0){
6        return false;
7    }
8
9    /* Search the lowest bid above the
10      minimum threshold */
11   for (uint i = 0; i < bidderUsersCount; i++) {
12     if(bids[bidderUsers[i]] >= minThreshold){
13       if(foundBid && bids[bidderUsers[i]] <
             bestBid){
14           bestBid = bids[bidderUsers[i]];
15           bestBidder = bidderUsers[i];
16       } else if (!foundBid){
17           bestBid = bids[bidderUsers[i]];
18           bestBidder = bidderUsers[i];
19           foundBid = true;
20       }
21     }
22   }
23
24   // Is the bid below the maximum threshold?
25   if(foundBid && bestBid <= maxThreshold){
26     // We have a winner!
27     hasWinner = true;
28     return true;
29   }
30   return false;
31 }
```

Figure 7: SC function to identify the winner of the auction.

function can be invoked only by the AmI system, which is the owner of the SC (line 5). The *unlockPayment()* triggers an effective ETH transaction (line 15) only when the auction yields a legitimate winner (line 8). Since, it is possible that the best bid requires a payment lower than the current ETH balance, at the end of the transaction, the residual amount of ETHs is transferred again back to the AmI system account (line 19).

VI. Conclusions and Future Work

In order to achieve a fully autonomic behavior, modern AmI systems should be able to negotiate services with unknown entities without the brokering of trusted third parties. A relevant scenario in which this requirement is particularly worthy of attention is the system self-maintenance.

In this paper we showed as smart contracts and blockchains can be exploited to create smart auctions through the Ethereum network. According to the proposed architecture, the AmI system is able to maintain an explicit model of itself that can be used to detect faulty devices, and to plan the actions to perform in order to get the system functioning again. These tasks are accomplished by two components, namely the *perception* and *planning* subsystems, that are based on a rule-based inference engine and a probabilistic decision-making model respectively.

Then, a subsystem responsible for creating and managing the smart contracts has been presented. The *smart contract handler* directly interacts with the Ethereum network and invokes all the functions of the smart contract that allow to manage the auction, identify the winner, and complete the payment.

```
1  function unlockPayment() public {
2
3    /*  Only the beneficiary of the repair
4        service can unlock the payment */
5    require(msg.sender == owner);
6
7    //  Has the winner been selected?
8    require(hasWinner);
9
10   /*  Check if the contract balance is
11       greater than the amount to be paid */
12   require(address(this).balance >= bestBid);
13
14   // Payment execution
15   bestBidder.transfer(bestBid);
16
17   /* Return of any remaining money in
18      the contract account */
19   owner.transfer(address(this).balance);
20 }
```

Figure 8: SC function to transfer payment to the winner of the auction.

The case study addressing the self-management of faulty smart home devices showed the feasibility of the proposed solution. Then, we aim to investigate how our architecture can be extended to support automatic interactions between AmI systems and other external service providers, e.g., energy providers and prosumers operating in a smart-grid. In such a context, a greater number of complex constraints should be considered in the smart contract, and this will be subject of future work.

References

[1] D. J. Cook, J. C. Augusto, and V. R. Jakkula, "Ambient Intelligence: technologies, applications, and opportunities," *Pervasive and Mobile Computing*, vol. 5, no. 4, pp. 277–298, 2009.

[2] E. Aarts and B. De Ruyter, "New research perspectives on Ambient Intelligence," *Journal of Ambient Intelligence and Smart Environments*, vol. 1, no. 1, pp. 5–14, 2009.

[3] A. De Paola, G. Lo Re, M. Morana, and M. Ortolani, "Smartbuildings: an AmI system for energy efficiency," in *2015 Sustainable Internet and ICT for Sustainability (SustainIT)*. IEEE, 2015, pp. 1–7.

[4] D. I. Tapia, J. A. Fraile, S. Rodríguez, R. S. Alonso, and J. M. Corchado, "Integrating hardware agents into an enhanced multi-agent architecture for Ambient Intelligence systems," *Information Sciences*, vol. 222, pp. 47–65, 2013.

[5] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of Things: vision, applications and research challenges," *Ad hoc networks*, vol. 10, no. 7, pp. 1497–1516, 2012.

[6] A. De Paola, P. Ferraro, G. Lo Re, M. Morana, and M. Ortolani, "A fog-based hybrid intelligent system for energy saving in smart buildings," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–15, 2019.

[7] M. Bartoletti and L. Pompianu, "An empirical analysis of smart contracts: platforms, applications, and design patterns," in *Proc. of the International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 494–509.

[8] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997.

[9] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform, 2013," *http://ethereum.org/ethereum.html*, 2017.
[10] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *https://nakamotoinstitute.org/bitcoin*, 2008.
[11] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.
[12] M. Iansiti and K. R. Lakhani, "The truth about blockchain," *Harvard Business Review*, vol. 95, no. 1, pp. 118–127, 2017.
[13] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: architecture, consensus, and future trends," in *Proc. of the 2017 IEEE International Congress on Big Data (BigData Congress)*. IEEE, 2017, pp. 557–564.
[14] P. McCorry, S. F. Shahandashti, and F. Hao, "A smart contract for boardroom voting with maximum voter privacy," in *Proc. of the International Conference on Financial Cryptography and Data Security*, 2017, pp. 357–375.
[15] T. Nugent, D. Upton, and M. Cimpoesu, "Improving data transparency in clinical trials using blockchain smart contracts," *F1000Research*, vol. 5, pp. 2541–2541, 2016.
[16] A. Hahn, R. Singh, C. Liu, and S. Chen, "Smart contract-based campus demonstration of decentralized transactive energy auctions," in *Proc. of the 2017 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, 2017, pp. 1–5.
[17] S. Huh, S. Cho, and S. Kim, "Managing IoT devices using blockchain platform," in *2017 19th international conference on advanced communication technology (ICACT)*. IEEE, 2017, pp. 464–467.
[18] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "Blockchain for IoT security and privacy: The case study of a smart home," in *Proc. of the 2017 IEEE international conference on pervasive computing and communications workshops (PerCom workshops)*. IEEE, 2017, pp. 618–623.
[19] M. T. Hammi, B. Hammi, P. Bellot, and A. Serhrouchni, "Bubbles of Trust: A decentralized blockchain-based authentication system for IoT," *Computers & Security*, vol. 78, pp. 126 – 142, 2018.
[20] S. Gaglio and G. Lo Re, *Advances onto the Internet of Things*. Springer, 2014, vol. 349.
[21] A. De Paola, S. Gaglio, G. Lo Re, and M. Ortolani, "Sensor9k: A testbed for designing and experimenting with WSN-based Ambient Intelligence applications," *Pervasive and Mobile Computing*, vol. 8, no. 3, pp. 448–466, 2012.
[22] M. Parashar and S. Hariri, *Autonomic computing: concepts, infrastructure, and applications*. CRC press, 2018.
[23] A. De Paola, "An ontology-based autonomic system for Ambient Intelligence scenarios," in *Advances onto the Internet of Things*. Springer, 2014, pp. 1–17.
[24] A. De Paola, P. Ferraro, S. Gaglio, and G. Lo Re, "Autonomic behaviors in an Ambient Intelligence system," in *2014 IEEE Symposium on Computational Intelligence for Human-like Intelligence (CIHLI)*. IEEE, 2014, pp. 1–8.
[25] E. Friedman, *Jess in action: rule-based systems in Java*. Manning Publications Co., 2003.
[26] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.
[27] C. Crapanzano, F. Milazzo, A. De Paola, and G. Lo Re, "Reputation management for distributed service-oriented architectures," in *Proc. of the4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop, SASOW 2010*, 2010, pp. 160–165.
[28] V. Agate, A. De Paola, G. Lo Re, and M. Morana, "A simulation framework for evaluating distributed reputation management systems," in *Proc. of the 13th International Conference on Distributed Computing and Artificial Intelligence, DCAI 2016*, vol. 474, 2016, pp. 247–254.
[29] V. Tron and F. Lange, "Geth. available from: https://github.com/ethereum/go-ethereum/wiki/geth," 2017.