Data quality control in blockchain applications*

Cinzia Cappiello[1], Marco Comuzzi[2], Florian Daniel[1], and Giovanni Meroni[1]

[1] Politecnico di Milano, Milan, Italy
[2] Ulsan National Institute of Science and Technology, Ulsan, Republic of Korea
{cinzia.cappiello,florian.daniel,giovanni.meroni}@polimi.it, mcomuzzi@unist.ac.kr

Abstract. This paper discusses the problem of data quality in blockchain applications at three levels of abstraction, i.e., conceptual, logical and physical. Conceptually, it makes explicit the need for information of typical data quality metrics for their online assessment. Logically, it analyzes how the adoption of blockchain technology affects the availability of the data needed for quality assessment. Physically, it identifies a set of implementation options that take into account the information needs of metrics and the restrictions by the technology; special attention at this level is paid to Ethereum and Solidity. Two case studies put the identified patterns and abstractions into context and showcase their importance in real-world distributed applications and processes.

Keywords: Blockchain, data quality, smart contracts, Ethereum, Solidity

1 Introduction

A *blockchain* is a distributed ledger, that is, a log of transactions that provides for persistency and verifiability of transactions [12]. *Transactions* are cryptographically signed instructions constructed by a user of the blockchain [15] and directed toward other parties in the blockchain network, for example the transfer of cryptocurrency from one account to another. A transaction typically contains a pre-defined set of metadata and an optional payload. Transactions are grouped into so-called *blocks*; blocks are concatenated chronologically. A new block is added to the blockchain using a hash computed over the last block as connecting link. A *consensus protocol* enables the nodes of the blockchain network to create trust in the state of the log and makes blockchains inherently resistant to tampering [10]. *Smart contracts* [13] extend a blockchain's functionality from storing transactions to performing also computations, for example to decide whether to release a given amount of cryptocurrency upon the satisfaction of a condition agreed on by two partners.

* This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program(IITP-2018-0-01441) supervised by the IITP (Institute for Information & communications Technology Promotion) and by the DITAS project funded by the European Union's Horizon 2020 research and innovation programme under grant agreement RIA 731945.

After having emerged as the core technology for cryptocurrencies, blockchains are increasingly adopted as building blocks for information system implementation. In particular, in the context of inter-organisational business processes, they can be used to create a trusted repository of transactions executed among a set of parties that do not necessarily trust each other. When underpinning systems supporting inter-organisational business processes, the quality of the data stored in blockchains becomes particularly critical. While data quality is trivial to enforce on transactions representing exchanges of cryptocurrency (e.g., the value transferred cannot be missing from a transaction and it must not exceed the amount of currency that the originator of a transaction owns), assessing data quality may become elaborated when transactions represent business interactions among parties collaborating in the context of a complex business process.

Given their nature, blockchains natively already provide for some quality guarantees regarding the data stored on them: the use of hashes to link blocks prevents tampering with data, while the use of cryptographic signatures provides for provenance and non-repudiation. However, blockchains also come with severe limitations that hamper assessing the quality of data stored on them, i.e., of the payload of transactions. This, in fact, is not subject to analysis and approval by standard consensus protocols and is treated by blockchains like a black box.

Analyzing these data requires either extending the internal logic of the underlying *consensus protocol*, or implementing data quality assessment logic, i.e., the data quality *controls*, in the form of suitable *smart contracts* to be invoked when needed. The former approach makes the whole blockchain aware of the content of payloads, but it also produces a blockchain infrastructure that is tailored to and restricted by the specific quality controls implemented. That is, all transactions would have to comply with the chosen payload formatting convention or the consensus protocol would not be able to process them correctly. As a consequence, this solution would suit only limited, private blockchain scenarios, in which only selected (and informed) nodes can participate to the network. The use of smart contracts, instead, enables the implementation of multiple data quality controls on top of generic blockchain infrastructures and their flexible, domain-specific use by applications. Yet, smart contracts, too, are subject to strict limitations that distinguish them from generic software modules:

- Smart contracts implement *passive application logic*; that is, they must be invoked by a client to be enacted and able to process data. This means that smart contracts cannot be implemented as listeners that automatically react to the presence of given data items inside the payload of generic transactions;
- Smart contracts cannot directly *query the blockchain* to retrieve data stored in transactions recorded on it; they only have access to the payload of those transactions explicitly directed to them as addressees, to data stored in their own, local variables, or to data held by other smart contracts and made available through suitable functions;
- Smart contracts cannot access *data outside the blockchain*. In order to guarantee the repeatability of the computations implemented by them, smart contracts cannot query external databases or Web APIs, as these might

produce different results in different instants of time. In order to obtain data from the outside world, so-called oracle smart contracts (short "oracles") can be used, which enable external data sources to push data into the blockchain, e.g., upon explicit solicitation or periodically – however, using standard transactions that are recorded on the blockchain;
- Executing smart contracts has a *cost*. Invoking a smart contract means generating a transaction directed to the smart contract and sending possible input data in its payload. This transaction is distributed over all nodes of the blockchain network and is subject to consensus, which consumes computing resources that need to be payed for. Also saving data on the blockchain consumes storage space that has a cost. Storing large amounts of data on the blockchain is thus not advisable, if not prohibitively costly.

In this paper, we aim to assist developers in the design and implementation of blockchain applications that come with their own data quality control requirements. Each application may have its own, domain-specific rules and conventions that need to be supported. The extension of existing applications with data quality controls is out of the scope of this paper, as extending existing applications is generally not possible: once data or code are written on the blockchain, they cannot be modified any longer. Hence, there is little space for improvement, and it is generally easier to just deploy a new application or a new version of it to guarantee specific data quality levels. This paper thus makes the following contributions:

- It identifies four combinable *conceptual patterns* representing the information needs of typical data quality controls for standard data quality metrics and proposes a set of *policies* for handling situations where these controls detect critically low data quality;
- It studies the four patterns in the context of blockchain technology and provides a set of *logical and physical implementation alternatives*;
- It shows the applicability of the identified solutions in the context of two *application case studies* with original data quality control requirements.

The remainder of the paper is organised as follows. Section 2 introduces the problem of data quality assessment in software applications. Section 3 discusses the proposed patterns and contextualizes them to blockchain systems. Section 4 shows the applicability of the proposed options in two application scenarios, while conclusions are drawn in Section 6 after discussing related work.

2 Data Quality Control Requirements

This section provides a *conceptual* discussion of information needs for the assessment of typical data quality dimensions and possible reaction in response to quality issues. Note that, in this paper, we focus on *on-line* data quality controls, that is, assessing the quality of data as they are submitted to an information system by a client application. We do not consider the case of off-line quality control, such as checking the quality of data stored in a system periodically or upon request.

2.1 Information needs for data quality assessment

Data quality is often defined as data *fitness for use* [11] and, as such, it is captured by a large number of data quality dimensions, the relevance of which depends on the application context. In order to provide a focused discussion, this paper considers a limited number of data quality dimensions, i.e., accuracy, completeness, consistency and precision. The former three dimensions are considered relevant in the context of traditional information systems development. The latter is particularly relevant in Internet of Things (IoT) scenarios, in which data may be provided continuously to a system by sensors.

Accuracy is defined as a measure of the proximity of a data value v to some other value v' that is considered correct [11]. Operationally, there are different ways of defining the accuracy of a value v depending on the nature of the domain of v. In data streams, accuracy is often analyzed with precision, that is, the degree to which consecutive measurements or calculations show the same or similar results. *Precision* is often defined in terms of the standard deviation of the measured values. The smaller the standard deviation, the higher the precision. *Completeness* is defined as the degree with which a given data collection includes the data describing the corresponding set of real-world objects [4]. *Consistency* refers to the satisfaction of semantic rules defined over a set of data items [4].

Each data quality dimension may have one or more metrics that specify how it can be calculated. The definition of the data quality assessment algorithms depends on the type of sources and on the type of data and may require additional metadata or rules, such as consistency rules, or expected values v' when assessing accuracy. Focusing on information needs, i.e., additional data required to assess the quality of a variable value, Fig. 1 depicts four situations that may occur.

Figure 1(a) refers to the situation in which the evaluation of the quality does not require additional information and therefore it can be conducted by considering only the analyzed value. For example, the accuracy of a value can be assessed by considering a specified business rule, using constants, such as "a temperature value is accurate if it is between 18 and 22 Celsius degree." Moreover, completeness is usually assessed by considering only the analysed value, i.e., checking whether the value is present or missing.

Figure 1(b) refers to the situation in which the evaluation of the quality of a value relies on the availability of one or more values of the same variable registered in the past. For example, a temperature value registered by a sensor may be considered accurate only if it does not exceed the average of values registered in the past 3 hours by more than 25%. Moreover, in an IoT scenario, assessing the precision of a sensed value needs always to consider the results of the previous measurements.

Figure 1(c) refers to the situation in which the evaluation of the quality of a value relies on single values of a number of other variables. For example, the accuracy of a patient name may be checked against the values of names and social security numbers provided by a public government registry, or the consistency of a temperature value registered by a sensor may be assessed against values of other variables registered by other sensors, such as pressure and relative humidity.

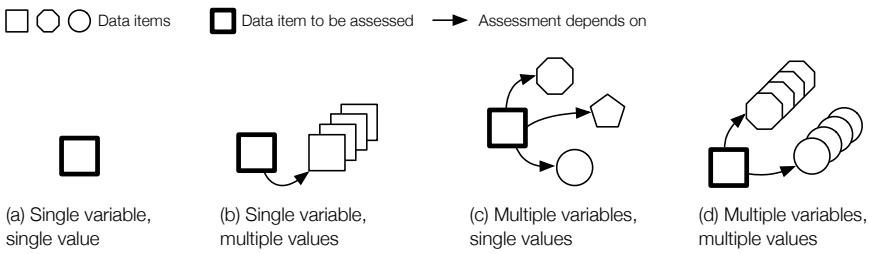

Fig. 1. Dependencies among variables and values when assessing data quality: single variable, single value vs. multiple variables and multiple values (history).

Finally, Figure 1(d) refers to the general case in which the evaluation of a value relies on multiple values of any number of other variables, possibly including the values of the variable which is being assessed. For example, a temperature value may be checked for consistency against historical values of temperature and pressure.

The next sections will show the way in which these different information needs may affect the implementation of data quality control in blockchains.

2.2 Quality control policies

Computing data quality measures on the fly allows an application to verify quality requirements at runtime, e.g., violations of consistency rules. This enables the implementation of data quality controls, if suitable reactions able to deal with identified issues are implemented. There may be different types of reactions in response to identified issues; deciding how to react is again an application-specific decision. In general, we distinguish five *policies* that may be adopted:

- *Accept value*: Sometimes, even though there is a clear violation of some data quality requirements, it may just be easiest to accept a value and just do nothing else. For instance, during the configuration of a sensor sending data to an information system, we may already know that the values communicated by the sensor during the configuration are not relevant to the system and hence, since they are not reliable, quality alerts can be ignored.
- *Do not accept value*: A possible decision may be to reject a low quality value and not write it into the system. For instance, in the case of sensor readings, this policy may apply when accuracy of data is important, that is, it may be preferable to have only highly accurate sensor readings instead of a complete series of readings of possibly low quality.
- *Log violation*: In some cases, it may be necessary to accept a value while, at the same time, flagging it of low quality. The flag may be considered by other applications in future computations. For example, if a social security

number provided by a patient does not match any record in a citizen registry, the system may be configured to accept it anyway, but with a flag to signal that a default quality control against a citizen registry has failed.

- *Raise event*: When a low quality value represents a critical situation that requires immediate reaction by an application or human actor, it may be necessary for a system to raise an event to notify someone or some other system. For example, low quality sensor readings may signal potentially critical issues when they concern an airtight container of dangerous goods being transported on public grounds.
- *Defer decision*: Finally, sometimes one single violation may not be enough to take a definite decision on how to intervene. In these cases, it may be an option to simply defer the decision for later re-evaluation.

Which of these policies are best depends on the application's data quality requirements, data retention obligations, expected reaction times, and similar. Each variable equipped with a data quality control may ask for a different policy. Policies may change during runtime, e.g., to react to different modes of execution, such as configuration vs. production. Ideally, the quality controls allow the application to dynamically switch or reconfigure policies as needed.

3 Data Quality Control on the Blockchain

Based on the analysis of information needs for data quality assessment provided in the previous section, we now discuss how data quality controls and reaction policies can be *logically* implemented in blockchains. Our assumptions underlying the rest of this paper are:

- Data quality controls are implemented using *smart contracts*, which provide the necessary flexibility to accommodate different quality control. We consider the extension of the consensus protocol, although feasible, not suitable to support application-specific quality controls.
- We focus on *on-line* data quality controls, i.e., on assessing the quality of data as they are added to the blockchain by a client via a suitable transaction and are to be stored by a smart contract.
- Client transactions always address a smart contract containing *application logic* of the distributed application that we want to equip with data quality controls. Transactions between parties without the involvement of a smart contract cannot be monitored from the inside of the blockchain, e.g., to prevent low quality data to be written.
- We require developers to identify which of their data need quality control and, for those that do, to delegate quality control to external, *quality-aware smart contracts* with suitable setters, getters and quality control logic. Developers thus must cede control of some of their variables, however in exchange for readily available and reusable quality control logics.

Before discussing implementation options, we discuss in the next section blockchain-specific patterns to capture the information needs for data quality assessment introduced in the previous section.

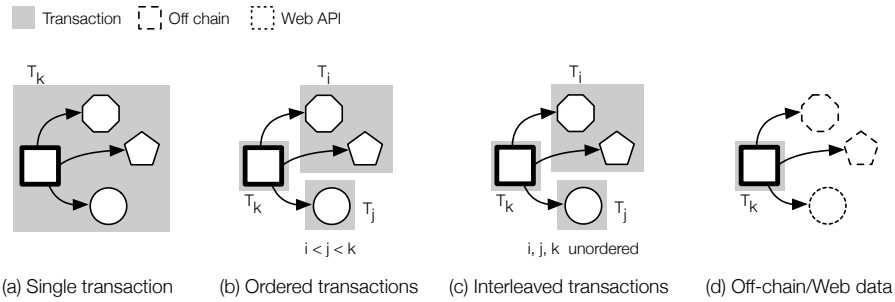

Fig. 2. Availability and correlation of data for quality assessment: transactions, transaction order, and on-chain/off-chain data. The figure uses the multiple variables / single values configuration for presentation purpose; other configurations are similar.

3.1 Data access for quality assessment in blockchains

Figure 1 introduced the four core types of data needs that may arise when assessing any of the discussed data quality metrics. The highlighted dependencies are conceptual and, especially in the context of blockchain applications, require a proper technological grounding for a developer to understand how to implement concrete quality metrics. The fundamental question that must be answered is *when* and *where* each of the necessary data items is available for processing.

Figure 2 summarizes the situations that we may encounter in a blockchain application, proposing four patterns that may be combined in function of the application's requirements. Data items (values of variables) required to assess the quality of a specific variable value may be available for processing via:

- *A single transaction* carrying all necessary values in its payload. This represents the easiest situation in which all necessary values are grouped and synchronized. A quality metric for the value of interest involving the other values, e.g., satisfaction of consistency rules, can be computed instantly as soon as the smart contract receives the payload.
- *Multiple ordered transactions* each carrying a piece of the information needed to compute a quality metric, with the value we want to compute the metric for always being the last to arrive. This requires collecting data from multiple transactions, but as soon as the value of interest arrives, the metric can be computed. An example could be checking the completeness of prior activities when reaching special milestone activities in a business process. Note that, owing to the distributed nature of blockchains, we cannot assume that transactions are received by all nodes in a network in the same order. However, this assumption may hold true in many practical cases if these transactions are sufficiently spaced in time.
- *Multiple interleaved transactions* where each transaction carries a piece of the information needed or the value of interest, but no specific order is guaranteed. Correctly computing a metric, in this case, requires correlating

transactions and waiting till all correlated values have successfully been recorded by a smart contract. For example, it may be necessary to wait for multiple recommendation letters before assessing a job application.
- *External data sources*, such as off-chain data stored by one of the nodes of the blockchain network or web-accessible data. As explained earlier, data outside the blockchain require help from so-called oracles in order to push data from the outside into the blockchain. This of course complicates the computation of metrics, requiring the involvement of external actors, and increases cost. An example for this pattern is the evaluation of the precision of a given value as a function of its historical values stored off-chain.

Like for the patterns of the conceptual dependencies among data items in Figure 1, also the above configurations may be arbitrarily combined in the context of specific applications to be developed. For instance, there may be a metric that requires both on-chain and off-chain data, or one that requires values that are distributed over multiple ordered transactions like a time series of values, e.g., temperature readings. Each of these combinations may thus require purposefully designed implementations of the respective quality metrics. Once implemented, their online application enables computing quality measures on the fly, discarding transactions, adding flags or raising events if needed, as specified by the relative quality control policies. Low quality flags or events raised, in turn, may trigger reactions according to application-specific policies.

3.2 Quality-aware smart contracts: implementation options

The previous discussion provided a condensed view of logical considerations that must be made so as to correctly implement data quality controls in the blockchain using smart contracts. We have seen that some of the characteristics that make blockchain technology and smart contracts powerful in the first place, however, pose severe restrictions on how quality controls can be implemented – a task that typically does not provide major issues off-chain.

Based on the data access patterns introduced in Figure 2 and our analysis of smart contract reuse options [7], we identify four core smart contract *implementation patterns* for quality controls:

(a) *Stateless smart contract:* if all data items that are needed to assess a given value are present in the payload of the same transaction, a simple, stateless smart contract with one function accepting the data as input is enough to implement the necessary control logic. Two sub-options are available:
  - *Ad-hoc contract*: we can implement a contract that accepts as input the values to be checked, implements the quality control logic, and responds with a respective assessment. In Solidity, this type of contract would be invoked by an application's smart contract using a standard *message call*.
  - *Reusable library:* we can also opt for the implementation of a so-called library, such as SafeMath (`https://bit.ly/2MRElXl`), that does not require the explicit invocation of an external contract. The application

```
contract HistoryDQContract {
    uint16[10] vars;  // array holding values subject to quality control
    uint8      index; // index of most recent value stored

    function set(uint16 _var) public returns (int8){
        if(index == vars.size) index = 0;   // update index
            else index++;
        vars[index] = _var; // store value, possibly overwriting oldest one
        return check();
    }

    function get() public return (uint16) { // fetch latest value
        return vars[index];
    }

    function getHistory() public return (uint16[]) { // fetch full history
        uint16[10] result; // to hold chronologically ordered values
        for (int8 i=index; i>=0; i--)
            result[index - i] = vars[i];
        for (int8 i=vars.size-1; i>index; i--)
            result[index + vars.size - i] = vars[i];
        return result;
    }

    function check() returns (int8){
        ... // TODO: implementation of quality control logic over full array
    }
}
```

Fig. 3. Solidity code fragment for single variable / multiple values quality controls based on a history of 10 values with no event raised upon detection of quality problems.

smart contract could attach the library to the data types to be controlled using the Solidity command `using` *library_name* `for` *data_type*. This could for instance guarantee that no unwanted values are ever written into a variable of those types. Doing so means transparently invoking quality controls using *delegate calls*.

(b) *Stateful smart contract:* if the quality control to be implemented instead asks for values stemming from different, ordered transactions, we need a stateful smart contract with one or more functions able to provide for the persistent storage of values across different invocations. As storage on the blockchain typically incurs a high cost, the objective should be to keep the data stored on-chain as small as possible. Again, there are two sub-options approaches:
   - *Multi-variable contract:* in this case, only single values of different variables need to be stored persistently, and we know that as soon as the value to be assessed arrives, all other values are up to date. Ideally each variable is equipped with suitable setters and getters to be used by the application's smart contract, while the setter of the variable whose quality is to be controlled also implements the respective quality control logic to be evaluated at each invocation.
   - *History contract:* in this case, also a history of values, e.g., using a simple array, by one or more of the variables stored in the quality control smart

```
contract FlaggingDQContract {
    uint16 varA;        // monitored variable
    bool   isUpdatedA;  // update flag
    uint32 varB;        // variable the control depends on
    bool   isUpdatedB;  // update flag

    function check() returns (int){
      if (isUpdatedA && isUpdatedB) { // if both variables are up to date
          isUpdatedA = isUpdatedB = false; // reset flags
          ... // TODO: apply quality control logic and return result
      } else return -1; // return if check not applicable yet
    }

    function setA(uint _varA) public returns (int){
      varA = _varA;
      isUpdatedA = true;
      return check(); // control quality if applicable
    }

    function setB(uint32 _varB) public returns (int){
      varB = _varB;
      isUpdatedB = true;
      return check(); // control quality if applicable
    }
    ... // TODO: implementation of getters
}
```

Fig. 4. Solidity code fragment for multiple variables / single value quality controls correlating two variables using simple Boolean flags.

contract must be maintained. In order to keep the cost of storage low, it is of utmost importance that the smart contract is properly configured so as to keep the history as short as possible without however compromising the evaluation of the quality metrics. Figure 3 illustrates a possible template for a history contract monitoring one variable.

(c) *Stateful smart contract + correlation:* here we do not have any guarantee that by the arrival of the value to be assessed all other values necessary for the assessment are up to date, as there may be interleaved transactions setting these values. This type of configuration thus asks for the correlation of values to decide when the quality assessment can be performed. We distinguish two distinct ways of correlating data depending on the correlation needs:

- *Flagging contract:* if the problem is correlating independent transactions where each time we only need the latest value of all variables involved, it may be enough to have flags that track which values have been updated since the last assessment. At each setting of a new value by the application, a function of the contract can be called implementing the actual quality control; if all values are flagged, the control is executed and the flags are reset, otherwise the control is deferred. The code fragment in Figure 4 provides an example of a flagging-based control.
- *Correlation contract:* if the contract is required to control multiple values of a given variable in function of respective sets of other values produced

```
contract OracleDQContract {
    OracleInterface oracle; // declaration of oracle contract
    uint var; // variable to be monitored

  event DQAssessmentDone(int result); // event for assessment notification

  function set(uint _var) public {
      var = _var;
      bytes4 sig = bytes4(keccak256("callback(uint)")); // set callback
      oracle.retrieveExtData(sig, this); // fetch external value
  }

  function callback(uint _extVar) public {
      emit DQAssessmentDone(check(_extVar)); // assess and notify result
  }

    function check(uint _extVar) returns (int){
        ... // TODO: implementation of quality control logic
  }
    ... // TODO: implementation of getter
}
```

Fig. 5. Solidity code fragment invoking oracle to fetch data for quality assessment. The contract emits an event `DQAssessmentDone` upon completed assessment. The assessment is executed asynchronously when the oracle invokes the `callback` function.

in independent but conceptually connected transactions (e.g., different business processes executed in parallel), it is necessary that the application itself provides additional metadata in the form of correlation identifiers for each new value that is set. This allows the quality control smart contract to properly correlate values as they arrive and to update independent counters for each value to be assessed. As soon as a counter reaches its target value, the respective value of interest can be assessed for quality and the counter reset again.

(d) *Smart contract + oracle:* in all those cases where there is a need for data from the outside of the blockchain, a simple smart contract is no longer enough. The help from a so-called oracle is needed, enabling the quality control smart contract to fetch data from the outside. In line with most of the literature, we distinguish two different types of data access requirements:
   - *Off-chain data:* these are data that are stored on one of the nodes of the blockchain network, e.g., in a database hosted by the node. Presumably, this node is thus aware of the application and is configured to push off-chain data into the blockchain (using transactions directed toward a smart contract of the application) either periodically or upon request (e.g., in response to an event risen by the application smart contract).
   - *Web-accessible data:* on the other hand, there may be the need for data that are not hosted by any of the nodes of the blockchain network and that are instead accessible via http calls over the Internet. In this case, the quality control smart contract may make use of an oracle smart contract to ask for external data, provide the oracle contract with a

callback function for the notification of the result, and wait for the oracle to raise an event to external observers able to provide the requested piece of information. This is the typical use of Provable (formerly Oraclize, `https://provable.xyz/`), the use of which is exemplified in Figure 5.

For those quality controls that raise an event that is meant to be intercepted by external actors for off-chain reactions, e.g., the re-calibration of a sensor, it may further be important to understand if the event was generated from the longest branch of the blockchain (the one that will survive) or from a fork. If the event is launched from a block included in a fork that eventually will be dismissed, that event may however already have been observed and processed by the external actor. Depending on the specific application's requirements, this may pose issues. To be on the safe side, it would be advisable to wait for the specific blockchain's minimum number of block confirmations to know if the event stems from the longest branch or not before taking action.

4 Application scenarios

4.1 Hazardous goods transportation

Let us consider the transportation of hazardous goods, such as liquids with a flash point of 23°C carried in special temperature- and pressure-controlled, watertight tanks. For safety reasons, it is typically further necessary to track live also the position of the tanks throughout the whole movement – the tanks are typically equipped with a suitable GPS transponder – to be able to trigger fast interventions by police or fire brigades in case of emergency. In order to prevent the parties involved in the transportation (there may be multiple carries) to alter or delete monitored data, e.g., to hide liability in case of an accident, we assume a blockchain is adopted to store monitored data and to record the movement.

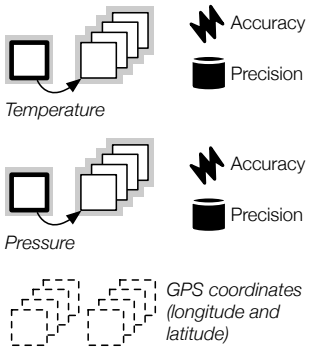

Fig. 6. Information needs, transactions and policies for hazardous goods monitoring.

The use of blockchain technology alone does not however prevent incorrect data to be stored in the first place. It may occur that one of the temperature sensors of the tank becomes defective, which could cause the reporting of inconsistent or inaccurate data. This in turn could lead to false alarms or, instead, to accidents that remain undetected. In the former case, the shipment might be stopped and emergency agencies deployed unnecessarily. In the latter case, emergency agencies might not be called in time. Either case would result in financial loss or damage.

To avoid these issues, mechanisms to evaluate the quality of sensor data before storing them permanently are required: the readings of both temperature and pressure can be checked for accuracy and precision before raising alerts. Accuracy can be checked against an interval of allowed values, e.g., temperature $T \in [0, 20]$,

and precision against a maximum standard deviation computed over a range of historical values, e.g., $stdev(T) < \Delta T_{max}$ over 5 readings; same for pressure. Let us assume, in this scenario, that violations of accuracy requirements are considered as more severe than excessive variations of precision. GPS coordinates are stored merely for documentation purpose and do not require any specific quality control.

In terms of the patterns identified in this paper, the described scenario involves four variables as summarized in Figure 6: temperature, pressure, longitude and latitude. Only the former two need to be stored on-chain and checked for quality; GPS coordinates can be kept off-chain. Accuracy is checked as soon as a value is available (synchronously, as proposed by the single transaction pattern in Figure 2(a) for single variable / single value dependencies); precision is computed based on a history of 5 values (pattern 2(b)). In terms of implementation, all checks can be implemented using a single smart contract of type *history* with support for multiple variables / multiple values for both temperature and pressure that raises alerts for accuracy violations and logs excessive deviations in precision (in line with the defined quality requirements).

4.2 Drug prescriptions

Each doctor (from general to specialized practitioners) prescribes, daily, dozens of medications to dozens of patients, each with different ongoing prescriptions and treatments. A typical error is the prescription of a drug that is incompatible with one already in use by a patient. This issue is particularly relevant for elderly people, who are more likely to need different medications to treat several chronic conditions. In order to prevent doctors from repudiating prescriptions, from tampering with their prescription record (e.g., to hide negligence) but also from false accusations by patients about treatments received or medications prescribed, a blockchain can be used to record prescriptions. To partly alleviate doctors from their burden we may want to implement a quality control that checks (i) if a prescription is correctly associated with a patient registered in a healthcare software system running off-chain and (ii) if the prescription is compatible with ongoing treatments by the patient (by consulting a suitable Web API provided by the Food and Drug Administration). This gives doctors a sense of responsibility but also helps them prevent errors.

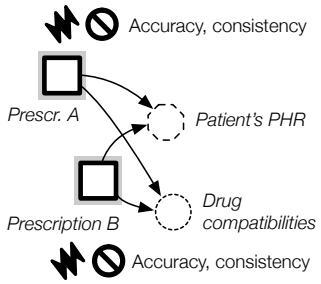

Fig. 7. Information needs, transactions and policies for drug prescription monitoring.

The described quality checks refer to a functional requirement of the application that can be seen as a data quality control and supported by the approach described in this paper. Given the use of both off-chain data (patient's personal health record) and web-accessible data (drug compatibility), the implementation of the quality control requires a smart contract that makes use of an oracle to fetch data from the outside for the evaluation of the consistency of a prescription,

which corresponds to pattern (c) in Figure 1 (multiple variables / single values). The quality check verifies the presence of the patient's social security number (SSN) in the off-chain system (accuracy) and the absence of incompatibilities among the drugs currently prescribed to the patient on-chain (consistency). Next to a function to store prescriptions for patients (each prescription consists of one SSN and one drug), the smart contract must further implement a function that allows the application to delete prescriptions at their natural termination or upon request to keep only ongoing prescriptions on-chain and save storage. The configuration of the resulting quality control contract is summarized in Figure 7, which can be easily implemented by extending the template in Figure 5.

5 Related Work

The introduction of second generation blockchains, i.e., with smart contract capability, has triggered researchers to analyze smart contracts, providing best practices for their software quality. For example, Atzei et al. [1] survey smart contracts deployed in the Ethereum blockchain, classifying their code vulnerabilities. Wohrer and Zdun [14] outline security patterns for smart contracts. Bartoletti and Pompianu [3] identify common programming patterns in Ethereum smart contracts and classify them based on the type of application. Zhang et al. [16] propose software patterns to ensure interoperability of smart contracts in the healthcare domain.

Concerning data quality, it is worth to notice that some contributions (e.g., [6]) claim that, since theoretically the information stored in the blockchain should be the exact representation of the events occurred in the real world, the data integrity and quality increases with the adoption of blockchain technology. In fact, the adoption of blockchain offers an automated means for creating, processing, storing and sharing information, therefore reducing human errors and improving the accuracy, completeness and accessibility of data supporting operational and decisional processes [8] [9]. For example, in [2], authors propose a medical record management system using blockchain technology. The authors claim that a benefit of the proposed system is the improved data quality and quantity for medical research. The availability of a greater data volume can be also a mean to compare data and correct errors, as described in [5]. This paper proposes, for the IoT scenario, a blockchain-based platform to assess and improve data quality of sensor data. However, none of the existing contributions provides a systematic approach to address data quality issues in blockchain.

6 Conclusions

This paper analyzes the issue of data quality in blockchain applications. Starting from a conceptual standpoint, in which we identify the information needs of data quality metrics, we propose a set of implementation options, focused on the Ethereum technology, to assist developers in crafting appropriate data quality controls in blockchain applications. Two case studies show the applicability of

the proposed approach. The approach does not yet consider in depth the relation between data quality controls and smart contract validation lifecycles. Given the probabilistic nature of proof-of-work consensus, values required for controlling data quality may be submitted in transactions ending up in dead end forks of the blockchain; this could create data completeness issues. Reactions to events raised by quality controls may be problematic if not properly analyzed before acting.

Since usage of the Ethereum network must be paid using gas, future work will analyse the computational and, therefore, economical, overhead introduced by data quality controls and how it can be minimised in the context of different implementation options. We will also consider the applicability of the proposed implementation options in other blockchain technologies besides Ethereum, such as Hyperledger Fabric and Sawtooth.

References

1. N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on ethereum smart contracts (sok). In *POST 2017*, pages 164–186. Springer, 2017.
2. A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman. Medrec: Using blockchain for medical data access and permission management. In *OBD 2016*, pages 25–30, 2016.
3. M. Bartoletti and L. Pompianu. An empirical analysis of smart contracts: Platforms, applications, and design patterns. In *FC 2017*, pages 494–509, 2017.
4. C. Batini and M. Scannapieco. *Data and Information Quality - Dimensions, Principles and Techniques*. Data-Centric Systems and Applications. Springer, 2016.
5. R. Casado-Vara, F. de la Prieta, J. Prieto, and J. M. Corchado. Blockchain framework for iot data quality via edge computing. In *BlockSys@SenSys 2018*, pages 19–24. ACM, 2018.
6. S. Chen, R. Shi, Z. Ren, J. Yan, Y. Shi, and J. Zhang. A blockchain-based supply chain quality management framework. In *ICEBE 2017*, pages 172–176, 2017.
7. F. Daniel and L. Guida. A service-oriented perspective on blockchain smart contracts. *IEEE Internet Computing*, 23(1):46–53, 2019.
8. C. Esposito, A. De Santis, G. Tortora, H. Chang, and K. R. Choo. Blockchain: A panacea for healthcare cloud-based data security and privacy? *IEEE Cloud Computing*, 5(1):31–37, Jan 2018.
9. S. Kar, V. Kasimsetty, S. Barlow, and S. Rao. Risk analysis of blockchain application for aerospace records management. In *AeroTech Americas*. SAE International, 2019.
10. D. Mingxiao, M. Xiaofeng, Z. Zhe, W. Xiangwei, and C. Qijun. A review on consensus algorithm of blockchain. In *SMC 2017*, pages 2567–2572. IEEE, 2017.
11. T. C. Redman. *Data quality for the information age*. Artech House, 1996.
12. N. Satoshi. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008.
13. N. Szabo. Smart contracts: building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought,(16)*, 1996.
14. M. Wohrer and U. Zdun. Smart contracts: security patterns in the ethereum ecosystem and solidity. In *IWBOSE@SANER 2018*, pages 2–8. IEEE, 2018.
15. G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.
16. P. Zhang, J. White, D. C. Schmidt, and G. Lenz. Applying software patterns to address interoperability in blockchain-based healthcare apps. *CoRR*, abs/1706.03700, 2017.