A Distributed Ledger approach to Digital Twin secure data sharing

Marietheres Dietz, Benedikt Putz, and Günther Pernul

University of Regensburg, Regensburg, Germany
{firstname.lastname}@ur.de

Abstract. The Digital Twin refers to a digital representation of any real-world counterpart allowing its management (from simple monitoring to autonomy). At the core of the concept lies the inclusion of the entire asset lifecycle. To enable all lifecycle parties to partake, the Digital Twin should provide a sharable data base. Thereby, integrity and confidentiality issues are pressing, turning security into a major requirement. However, given that the Digital Twin paradigm is still at an early stage, most works do not consider security yet. Distributed ledgers provide a novel technology for multi-party data sharing that emphasizes security features such as integrity. For this reason, we examine the applicability of distributed ledgers to secure Digital Twin data sharing. We contribute to current literature by identifying requirements for Digital Twin data sharing in order to overcome current infrastructural challenges. We furthermore propose a framework for secure Digital Twin data sharing based on Distributed Ledger Technology. A conclusive use case demonstrates requirements fulfillment and is followed by a critical discussion proposing avenues for future work.

Keywords: trust frameworks · distributed systems security · distributed ledger technology · digital twin.

1 Introduction

Hardly anything has revolutionized society as much as digitization. At its beginning, data from everyday life was captured and stored digitally. After reaching significant amounts of digital data, recent years have been devoted to gaining relevant insights into data by leveraging Big Data Analytics, Artificial Intelligence and so on. A next step in digitization is now emerging in the form of the Digital Twin (DT) paradigm.

The Digital Twin refers to a digital representation of any real-world counterpart, at most times an enterprise asset. Its core building blocks are asset-specific data items, often enhanced with semantic technologies and analysis/simulation environments to explore the real-world asset digitally. The DT thus allows management of such an asset ranging from simple monitoring to autonomy. An essential part of the concept is the inclusion of the whole asset lifecycle. To integrate all lifecycle participants, the DT should provide comprehensive networking for its data, allowing it to be shared and exchanged [4].

Although the DT concept certainly advances digitization, it nevertheless poses new challenges in terms of IT security, especially in industrial ecosystems [10,18]. Most notably, security must be maintained during the exchange of DT data between different, non-trusting parties. For instance, consider the DT of a power plant. Synchronizing tasks between twins should uphold integrity to avoid manipulated operations on the power plant. Also, involved parties should not be able to read every shared data element (e.g. the manufacturer of the power plant need not know the plant's current status), resulting in confidentiality requirements. To the best of our knowledge, current DT frameworks do not permit secure data sharing. Bridging this gap, our work provides a framework introducing security-by-design in DT data sharing.

To achieve this goal, we consider Distributed Ledger Technology (DLT). DLT is the umbrella term for distributed transaction-based systems, shared among several independent parties in a network. Distributed Ledgers have built-in mechanisms for access control and asset management, including authentication and authorization mechanisms. We focus on permissioned distributed ledgers, which target enterprise usage by restricting access to fixed set of independent and semi-trusted participants. One of the main reasons for using a Distributed Ledger is disintermediation, replacing the need for trust in a third party or central operator through a replicated and integrity-preserving database. Inherent transparency and auditability are additional advantages over centralized solutions. Due to these properties, DLT is uniquely suited to solve the challenges of DT secure data sharing.

Accordingly, this work proposes a framework for secure DT data sharing across an asset's lifecycle and collaborating parties based on DLT. We contribute to the body of knowledge by offering a solution without a trusted third party (TTP) based on security-by-design. The remainder of this paper is organized as follows: Chapter 2 introduces the background of our work. Afterwards, we proceed to the description of the current problems in DT data sharing and name the resulting requirements for secure DT data sharing (Chapter 3). In Chapter 4, we provide a framework for secure DT data sharing for multiple parties based on DLT. To show practical relevance and the functionality of our framework, a use case is provided in Chapter 5. In Chapter 6, we evaluate our approach in terms of fulfillment of the stated requirements. To conclude, Chapter 7 sums up the main contributions and gives an outlook for future work.

2 Background

At present, the *Digital Twin* phenomenon is still in its infancy. Nevertheless, implementation and design of this concept are addressed to date, especially in the area of Industry 4.0. With strong focus on the industrial domain, the major part of research suggests DT implementation through AutomationML-formatted descriptive data of the real-world counterpart, e.g. [20,2,6]. The XML-based AutomationML (AML) format describes industrial assets and provides object-orientation for modeling the asset's physical and logical components [20]. Eck-

hart and Ekelhart [6] propose a framework for using a DT's simulation mode for security purposes such as pen testing. While these works focus on an initial development of a DT, the consideration of data sharing functions are still missing. However, exchanging data is vital for enabling the lifecycle integration and collaboration [4]. Our work builds on existing DT propositions, resulting in a concept that can be applied in a complementary way to enable secure DT data sharing.

Regarding DT data sharing, both the communication between lifecycle parties and the bidirectional communication between the DT and its real-world asset counterpart need to be considered. Bidirectional communication consists of the DT's instructions for the asset and the asset's status update for the DT. To uphold integrity among multi-domain DT models, Talkhestani et al. [21] offer a solution. They detect model changes by applying anchor points, and upon detection synchronize the DT while keeping model dependencies consistent. However, this includes drawbacks such as the manual creation of anchor points and reliance on a Product Lifecycle Management (PLM) system, while our solution offers platform-independence. Security aspects, such as the guarantee for all lifecycle partners to access the data while upholding confidentiality, are not considered to date, but integrated in our solution.

DT management is a form of enterprise asset management, which is one of the prime use cases of Distributed Ledgers [1]. Distributed Ledgers are able to track events and provenance information along an asset's lifecycle and increase transparency for all participants. For example, Litke et al. [12] studied the benefits of Distributed Ledgers for different actors in supply chain asset management, a research area closely related to DT asset management. In another study, Meroni and Plebani [14] investigate how the blockchain technology can be used for process coordination among smart objects. Smart objects are similar to DTs in that they are applied for monitoring physical artifacts. An issue with their proposed approach is that sensor data is also stored on the blockchain, which can be detrimental to performance and scalability. We consider this issue and provide a solution to overcome this obstacle.

3 Problem statement

On the one hand, DTs should facilitate the access to asset information for different stakeholders along its lifecycle [17]. It is a task which enables feedback loops, while stepping towards a circular economy [3]. On the other hand, the involved parties do not necessarily trust each other, resulting in a confidentiality dilemma. A useful example is given in [13]: Two separate standalone DTs exist for a single device instance, one for the manufacturer and the other at the customer site – due to information security reasons. Additionally, current works state that enterprise infrastructures need to overcome the following obstacles to provide secure DT data sharing:

- application of different tools [24,13]
- usage of various data formats [13]

- missing standards [4]
- broken information flow across lifecycle phases [24,13]
- clarification of the ownership of information [13]

This calls for a holistic approach that provides confidentiality and integrity, two central security dimensions in networks [26].

3.1 Digital Twin Model

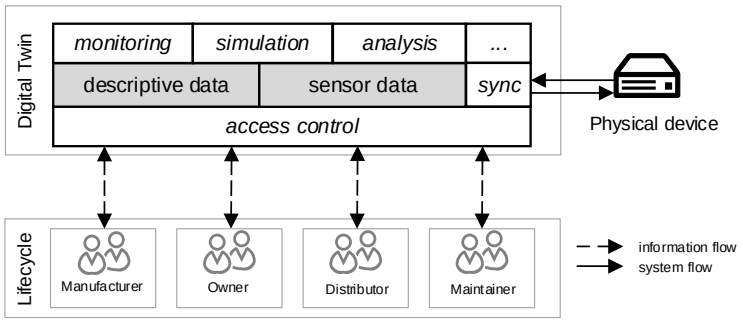

Fig. 1. Overview of the asset lifecycle participants interacting with the DT.

Figure 1 illustrates DT data sharing and an exemplary set of lifecycle stakeholders. The depicted DT model comprises different *capabilities* and two types of asset-specific data. *Descriptive data* refers to static properties of the device and infrequently changing state information. This data is mainly produced by users. *Sensor data* occurs frequently and should be available in near real-time. It is generated by sensors of the physical asset or in its proximity, which provide valuable information on the asset's environmental conditions. Moreover, data of both types needs to be synchronized with the physical counterpart. Therefore, the *sync* capability compares the state of the DT to its real-world counterpart and resolves possible discrepancies.

The *access control* capability provides authentication and authorization modules to enable data sharing of involved parties without hampering confidentiality. The *monitoring*, *simulation* and *analysis* capabilities represent advanced operations of the DT. Depending on the extent of the operations present in a DT, DT status data can be returned to the participant or the real-world counterpart's state can be modified.

The depicted information flows show how information about the physical device is gathered from and sent to the lifecycle parties. The system flows represent necessary bidirectional synchronization between the DT and its real-world counterpart as stated in Section 2. Both flows contribute to making the data sharing activities of the involved parties traceable. This enables feedback from the latest stages of the asset lifecycle to the earliest ones [17].

3.2 A formal basis for secure Digital Twin data sharing

Although a methodological literature analysis to establish requirements is the state-of-the-art approach, it is currently not sensible to carry out with regard to our research focus. On the one hand, this is due to the fact that only a small number of publications exist. In addition, data sharing has not yet been a focus in DT literature to date. Moreover, security-by-design concepts have not been considered yet. Therefore, we establish a formally valid basis in order to create a uniform understanding of DT data sharing. To derive the requirements, the

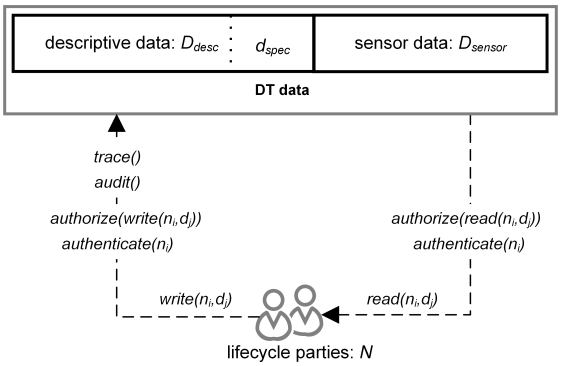

Fig. 2. Control flows for a single DT.

mechanisms to achieve the central goal of **secure DT data sharing** have to be examined in detail. Figure 2 illustrates the formal functions required to achieve this goal, which are also described hereafter.

DT Data: We see DT data twofold: At first, there is a set of descriptive data elements $D_{desc} \coloneqq \{d_1, ..., d_m\}$ varying from documents to models or analytic outcomes. Its essential data element is the specification of the DT $d_{spec} \in D_{desc}$. The second set contains environmental, device-produced data, namely sensor data $D_{sensor} \coloneqq \{d_1, ..., d_n\}$, whereby $D_{desc} \setminus D_{sensor}$.

Sharing: A finite set of lifecycle parties $N \coloneqq \{n_1, ..., n_k | k \geq 2\}$ can share the respective data elements of D_{desc} (write operation) or access the data elements of D_{desc}, D_{sensor} (read operation). This results in the following necessary functions: $write(n_i, d_j | d_j \in D_{desc})$ and $read(n_i, d_j | d_j \in D_{desc} \vee d_j \in D_{sensor})$.

Note that $1 \leq i \leq k$ as well as $j \begin{cases} 1 \leq j \leq m & \text{if } j \in D_{desc} \\ 1 \leq j \leq n & \text{if } j \in D_{sensor} \end{cases}$.

Security-by-design: Security-by-design infers introducing security mechanisms at the very beginning of a system's design [22]. In terms of DT data and sharing security, data integrity and confidentiality mechanisms are of special interest. Confidentiality in terms of securing data from view of non-trusted third parties can be reached by access control mechanisms [19]:

authentication:

$$authenticate(n_i)$$

authorization:

$$authorize(read(n_i, d_j))$$

$$authorize(write(n_i, d_j))$$

Integrity of data can be achieved by auditability and traceability of write operations. Given D_{desc} as the origin set of data, D'_{desc} is the set of data after a data element d_j is added to the origin set. The following functions can cover integrity aspects:

auditability:

$$audit() : D_{desc} \rightarrow D'_{desc} \iff write(n_i, d_j) \quad \wedge$$

$$D_{desc} \nrightarrow D'_{desc} \iff \neg write(n_i, d_j)$$

traceability:

$$trace() : D_{desc} \rightarrow D'_{desc} \implies D_{desc} \circ D'_{desc}$$

Thereby, auditability guarantees that D_{desc} is transformed to D'_{desc} in case of an authorized write operation whereas other operations are not able to transform the data in any way. Traceability ensures that authorized writes of data elements and thus, transformations of D_{desc} to D'_{desc}, are chained up. In conclusion, data integrity is ensured as the data cannot be manipulated or tampered with in retrospect.

3.3 Requirements for secure DT data sharing

To provide a sound solution for secure DT data sharing, the following requirements were derived from the formal basis and the aforementioned challenges identified in the literature analysis.

R1. Multi-party sharing. To enable lifecycle inclusion, a vital characteristic of the DT paradigm [4], the multiple stakeholders N involved in the lifecycle have to be considered. As described in Figure 1, parties can vary from manufacturer to maintainer. However, all involved parties are pre-registered and therefore determinable.

R2. Data variety support. At the heart of the DT lie the relevant digital artifacts D_{desc}, D_{sensor}, which vary from design and engineering data to operational data to behavioral descriptions [4]. Thus, different data types and data formats [13] need to be supported during data sharing. For instance, Schroeder et al. claim that using the semi-structured AutomationML format to model attributes related to the DT (d_{spec}) is very useful for DT data exchange [20]. In addition to semi-structured data, structured data (e.g. sensor tuples in D_{sensor}, database entries) and unstructured data such as human-readable documents can be asset-relevant and shared via the DT.

R3. Data velocity support. Often, DT data is distinguished between descriptive, rather static data, and behavioral, more dynamic data (see Figure 1). The latter changes with time along the lifecycle of the real-world counterpart [20]: With each lifecycle stage the asset-related information evolves, resulting in different versions and a dynamic information structure [17]. Naturally, dynamic data includes sensor data D_{sensor} – which mostly refers to the actual state of the real-world counterpart [8]. While the infrequently changing data D_{desc} might not require high throughput, sensor and dynamic data D_{sensor} accrues in intervals ranging from minutes to milliseconds. Therefore, the solution must support high throughput and low sharing latency for efficient sharing of dynamic data – thus supporting data velocity.

R4. Data integrity and confidentiality mechanisms. An important requirement is taking into account data security features, especially integrity and confidentiality. At first, this requirement aims at safeguarding data integrity to avoid wrong analytic decisions based on manipulated data. It can be ensured by *audit*() and *trace*() mechanisms. The second main security objective is to avoid confidentiality and trust problems while enabling multi-party participation. This calls for restricted data access dependent on the party through *authenticate*() and *authorize*() functions, while ideally keeping the effort for user registration low. Different levels of confidentiality should be possible for different data elements. For instance, D_{sensor} might need a lower level of protection than D_{desc}, as the latter might include sensitive corporate information such as blueprints. Detailed *authorize*() functions, providing access-restrictions for each data element, can cover this aspect.

R5. Read and write operations. To interact with DT data, a DT data sharing solution must provide *read*() and *write*() data operations for the sharing parties. The allowance of operation modes for the data elements should be chosen carefully for each party to ensure **R4** (cf. Figure 2).

Overall, we do not claim that these requirements are complete. There may be other requirements of importance, but regard these as essential for the following reasons. On the one hand, these requirements were found to be mentioned most often in the reviewed literature, while others were less frequently mentioned and are therefore considered of lower importance (see Section 6.2 for further explanation). On the other hand, the stated requirements were also the main focus in various practitioners reports (e.g. [16,25,9]) and during discussions with experts.

4 Solution architecture

In order to develop a framework for secure DT data sharing, we first evaluate the suitability of DLT in Section 4.1. Afterwards, Section 4.2 explains the system architecture and Section 4.3 explains how the various data types are stored. Section 4.4 details the inclusion of the DT capabilities as part of the DLT solution. Finally, Section 4.5 explains the initial setup procedure for our framework.

4.1 Technology selection

To develop a solution architecture, we first evaluate different data storage solutions' properties to select the technology best suited to fulfill the requirements.

A centralized solution could be created in the form of a portal, operated by a third party or the operator of the twinned device. This requires trust of the participating parties towards the portal maintainer, as the maintainer could manipulate data or revoke access to the DT for other parties. A distributed approach jointly operated by all participants could solve this trust issue. Distributed Ledgers represent such a distributed solution. They permit verifiable decentralized execution of business logic via *smart contracts*, ensuring that rules and processes agreed upon by the lifecycle participants are followed.

We evaluate the applicability of Distributed Ledgers to our DT data sharing requirements based on the blockchain applicability evaluation framework by Wüst and Gervais [28]. As illustrated in Figure 1, there is a need to store various types of data as part of the DT state. Multiple parties interact with the twin during its lifecycle who do not fully trust each other. These writers are usually known in advance or change infrequently (i.e. the maintenance service provider changes). These characteristics lead to the choice of a public or private permissioned blockchain in the framework [28]. In our case, this choice depends on whether public auditability is required or not. Since read-only public node access can be enabled during implementation, we do not prescribe either option here. We assume that for the majority of use cases the confidentiality requirement will lead to a private system.

4.2 System architecture

The proposed DLT-based architecture for secure DT data sharing is shown in Figure 3. Every participant runs three components: a node of a **Distributed Hash Table (DHT)**, a node of the **Distributed Ledger** and a **client application**. The DHT and Distributed Ledger make up the shared data storage, while the client application is responsible for the user interface and backend logic for retrieving and processing the data stored on the ledger and DHT. For owners of twinned physical devices, a **Device agent** manages the physical devices and coordinates their interactions with the system. As part of operational technology, the Device agent functions as a bridge between the cross-organizational asset management system and the physical devices controlled by a single organization.

Data storage systems based on distributed ledgers have two ways of storing data: on-chain and off-chain [29]. On-chain storage is restricted to transactions and the internal state storage of smart contracts. Due to full replication of on-chain data, items larger than a few kilobytes in size need to be stored in a different, off-chain location. Using a traditional database would however result in a single point of failure or reintroduce a trusted party.

For this reason, we resort to a structured DHT for large data items. DHTs are distributed key-value stores, where all key-value pairs are mapped to one or more nodes. The DHT entries can be linked to the corresponding on-chain asset

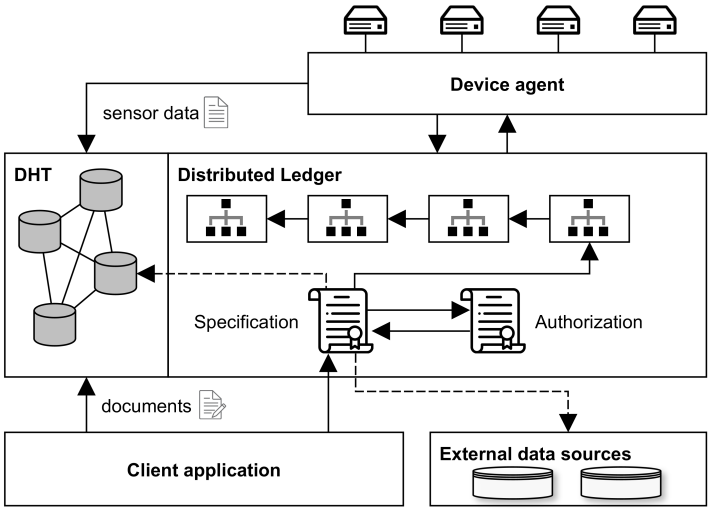

Fig. 3. DLT-based architecture for DT data sharing.

based on the DHT key hash. By storing the hash on the blockchain, integrity of the off-chain data can be verified after retrieving it from the DHT. To maintain confidentiality and availability, data stored on the DHT is encrypted, sharded and replicated. Correspondingly, an access control mechanism is needed to allow authorized parties to access the data. The k-rAC scheme illustrates how a DHT can implement the required functionality [11]. In k-rAC, access control is implemented using access control lists (ACL) stored along with each key-value pair on the DHT. We propose reusing the Distributed Ledger's public key identities for DHT authentication. A symmetric key is used for encryption, which is then available to authorized parties by encrypting it with their public key. The encrypted access keys are distributed with each data item's ACL. Manipulation of the ACL is prevented by requiring a quorum of $2k+1$ nodes for write operations, where k is the number of tolerated malicious nodes.

4.3 Data storage

There are two types of **descriptive data** that need to be stored by the system: a machine-readable specification and device-related unstructured data (i.e. human-readable documents). The **specification** includes a description of the device's hardware components as well as their functions. The DT's physical properties are derived from this specification. For our work we assume that AML is used to describe the physical asset. The AML specification is stored on the ledger in a modifiable way. This approach guarantees that updates to the device specification are observed by all parties. Distributed Ledgers can store complex modifiable state by using *smart contracts*. We thus refer to the resulting contract as the *specification contract*.

Unstructured data can be uploaded to the system and may subsequently be annotated or modified by other parties. Due to its size it cannot easily be parsed and stored in contracts. For this reason, it is stored off-chain and registered in the smart-contract with a document title and a hash of the contents. To update a document, a new version must be uploaded to the DHT and the smart contract reference updated. This ensures that changes to the documents are traceable.

Sensor data needs to be stored off-chain due to its frequent updates and the considerable amount of generated data. A history of the sensor data is kept to allow for further analysis, e.g. predictive maintenance or troubleshooting. The link to the on-chain data is established via a pointer to the off-chain storage location, stored on-chain in the specification contract. To avoid having to update the storage location hash every time new sensor data is uploaded to the DHT, we take advantage of *DHT feeds*. This concept is inspired by the Ethereum network's DHT Swarm [7]. In Swarm, a feed is defined by a feed manifest with a unique address on the network. The feed manifest's owner (i.e. the physical device) is the only user permitted to upload signed and timestamped data to this address. Any data format can be used and a history of uploaded data is kept. The DHT feed enables frequent sensor data sharing without having to update an on-chain reference. Based on the feed, the client application may compare sensor updates with expected values derived from the specification contract to detect anomalies. Additionally, there is no need for directly accessing the physical device, which may reside in a protected network. Instead, data updates are readily available on the DHT for authorized participants.

Many organizations also have additional internal data sources or microservices that provide structured data relevant to the Digital Twin. These data sources can be included in the twin by adding references (i.e. an URI) to the DT specification contract. This allows inclusion of legacy data sources and complex data which cannot easily be stored on a DHT (i.e. relational data). If the external data source requires authentication, it is the responsibility of the data source provider to ensure access rights for the DT ledger's identities.

Listing 1.1 shows a pseudocode representation of the data types stored in the specification contract. The syntax is inspired by Ethereum's Solidity smart contract programming language. All data stored on the contract is readable by all lifecycle participants. Besides general device metadata, the contract also includes a program call queue for interaction with the physical device's program interfaces (see also Section 4.4). Since smart contracts must be deterministic and thus cannot interact with files, the AML specification is stored in a string variable. This variable can later be parsed and modified, as illustrated in Section 4.4. Hash references to new original documents on the DHT are kept track of in the `documents` mapping. The hash serves as an identifier, while the `document` struct provides metadata. Updated versions of each document are stored in the `documentVersions` mapping. The componentID and corresponding feed reference of the sensor data stream on the DHT are stored in the `sensorFeeds` mapping.

```
/* metadata and specification*/
string deviceName
string deviceID
string deviceAML
string[] callProgramQueue

/* additional descriptive data */
struct Document {
  uint timestamp
  string description
  address owner
}

struct ExternalSource{
  string URI
  address owner
}

mapping(string=>Document) documents
mapping(string=>string[]) documentVersions
ExternalSource[] externalSources

/* sensor data */
mapping(string=>string) sensorFeeds
```

Listing 1.1. Data structures of the specification contract

```
/* descriptive data interfaces */
function addDocument(document)
function addDocumentVersion(string hash)
function removeDocument(string hash)

function addExternalSource(string URI)
function removeExternalSource(string URI)

/* sensor data interfaces */
function addSensorFeed(string componentID,
      string reference)

function removeSensorFeed(string componentID)

/* interaction with the specification */
function insertAML(string amlCode, string
      parentID, string afterID)
function removeAML(string ID)
function callProgram(string programName,
      string parameters[])
```

Listing 1.2. Function interfaces of the specification contract

4.4 Capabilities

We focus on the three capabilities required for accessing and publishing DT data: *DT interaction*, *access control* and *sync*.

DT interaction refers to the information flows in Figure 1, which allow users to interact with the twin's data. The specification contract implements this functionality. It allows users to read and potentially modify the DT instance. The relevant interfaces that can be called with transactions are shown in Listing 1.2. New or updated references to documents may be appended by any authorized user. The same applies to external data sources and sensor feed references to the DHT. The specification can be manipulated by inserting or removing specific AML segments, which are identified by their ID. To determine the position of a new AML code segment in the AML document, the parent ID and the ID of the preceding element need to be passed as parameters. The twin's program interfaces for setting device parameters can be accessed via `callProgram`. This function checks authorization, finds the requested program in the AML specification and places it in a queue for the Device agent to retrieve. The agent then forwards the program call to the device for execution.

The *access control* capability is responsible for authentication and authorization of user interactions with the DT data. For user authentication, accounts are created on the blockchain and represented by their public key. An initial solution could be provided by the framework's built-in identity management, for example Hyperledger Fabric's Membership Service Provider (MSP) [1]. The MSP lists the certificate authorities who may issue digital identities for the Distributed Ledger. The same identity can then be reused for authentication in

the DHT. Authorization is realized in a separate access control smart contract. Any protected interaction with the Digital Twin is first authorized through that contract. Such interactions are for example modifications of the twin's properties, like changing parameters or modifying its specification. A query from the client application provides an identity to the specification contract, which then interacts with the authorization contract to determine if the user is allowed to perform the action. Authorization is then granted or denied based upon a stored role-permission mapping. Accordingly, the contract's interfaces are based upon a Role-based Access Control (RBAC) scheme. We do not describe the access control contract in detail here, as there are other works describing blockchain-based access control schemes [5].

The *sync* capability requires regular interaction between the Device agent and the Distributed Ledger. For synchronization, the Device agent pulls updates from the real-world asset and uploads them to the off-chain DHT sensor data feed. The Device agent monitors the ledger and pushes any modifications instructed by committed on-chain transactions to the asset. The synchronization interval depends on the use case.

Other DT capabilities like *monitoring, simulation* and *analysis* can be executed off-chain by interacting with the local copy of the ledger. Simulation or analysis instructions and results can be shared on the ledger as documents. This would allow other parties to verify the results, should they desire to do so.

4.5 Setup Process

Initially, each lifecycle participant sets up one network node running both a DHT and a Distributed Ledger node. These serve as local replicas of ledger data and access points for off-chain data. They may also be used for transaction-based interaction with the smart contracts. Additionally, an identity provider must be set up to allow federated identities from all participating organizations based on public key certificates.

Once the network is set up, a Digital Twin instance can be created on the ledger by the device owner. The manufacturer should first provide the AML file to the owner, who then proceeds to set up a Digital Twin sharing instance on the ledger. The client application provides the interface to upload the file and create a smart contract based on it. Before uploading, the owner also needs to specify the access rights associated with the various parts of the specification. Although use case dependent, sensible default values could be *write* access by owner and maintainer and *read* access by everyone else.

In this way, any number of Digital Twin instances can be created by the various parties on the network. Each instance is represented by a specification contract. Subsequent modifications take place via authorized on-chain transactions and are stored as part of the contract's internal state. As a result, auditing the twin is possible by (actively or retroactively) monitoring smart contract transactions for anomalies.

5 Use Case

This chapter intends to show how the theoretical framework developed in Chapter 4 is traversed in a use case. To begin with, the overall setting of the use case is described in Section 5.1, while the subsequent Section 5.2 iterates the use case through the solution architecture. At last, a summary is given, focusing on the automation degree in data sharing and the reading operation (Section 5.3).

5.1 Setting

The setting is chosen close to reality. The asset, the real-world counterpart to the DT, is a bottling plant, where bottles are filled with beverages. The parties involved in the asset lifecycle are a manufacturer, an owner, a maintainer of the bottling plant and an external auditor that audits the safety of our bottling plant. For our use case, we consider the following scenario: The bottles are flooding due to a broken sensor in the bottling plant. Consequently, the maintainer detects the damage and changes the broken sensor in the bottling plant.

This entails the following shared data interactions. At first, the specification of the plant needs to be updated by replacing the broken sensor's specification entry with the newly added sensor. Additionally, the new sensor's data stream has to be integrated in place of the old sensor stream. Other documents concerning the maintenance task might also be shared, such as a maintenance report.

While the maintainer is the only party sharing data in this scenario, the owner should also be updated on the state of the bottling plant. Furthermore, the manufacturer needs to be informed that the sensor is broken, so that an analysis of the time and circumstances can be conducted. This way relevant insights for future plant manufacturing can be gained. Additionally, the external auditor needs to access the information about the maintenance task to review the procedure in terms of safety compliance.

5.2 Framework iteration

This use case triggers a specific logical order of events in the framework, which are highlighted in Figure 4 and described hereafter. The framework first comes into play when the maintainer replaces the broken sensor.

1. All devices are connected with the **Device agent**, which registers the exchange of the broken sensor. Additionally, it gathers information about the new sensor.
2. Following the new sensor connection, the **Device agent** forwards the new incoming data stream of the sensor into the **DHT**. The location of the stored sensor stream in the **DHT** is registered by the **Device agent**.
   The **Device agent** then sends a transaction containing the new sensor specification to the **Distributed Ledger**. This transaction invokes the specification contract, resulting in several updates. First, the old sensor entry is removed and the new sensor specification given by the **Device agent** is

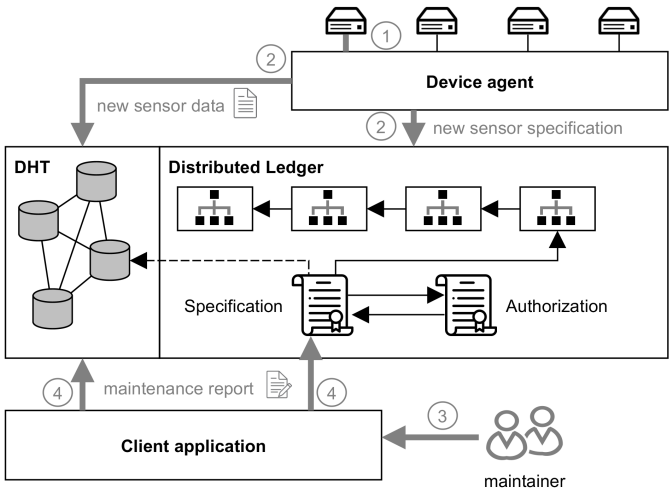

Fig. 4. Use case tailored architecture for DT data sharing.

added. Secondly, the storage location of the sensor stream on the **DHT** is added by a reference to the location. These three transactions concerning the specification are stored on the **Distributed Ledger**.

3. Having performed the maintenance task, the maintainer writes a maintenance report and pushes it onto the **Client application**.
4. The **Client application** adds the maintenance report by performing two actions. Firstly, it adds the report to the off-chain **DHT**. Secondly, it stores the reference to the **DHT** location of the report on the specification contract. Thereby, the location is added to the entry of the sensor specification.

5.3 Results

In a nutshell, the recognition of new sensor and the AML update with the new component is already accomplished by the **Device agent** without requiring human interaction. The new data stream is automatically forwarded to the **DHT** and the reference to the new storage location of the component's data stream is added to the specification contract. Additional unstructured non-specification data (e.g. the maintenance report) can be added manually. The **Client application** takes care of the necessary background work by inserting the file into the **DHT** and adding the respective storage reference into the specification contract.

All participating parties can view the latest transactions on the ledger – presented in a comprehensive way in the **Client application**. Advanced **Client applications** could also notify the user whenever an ledger update takes place.

Considering security, the advantages of this framework shine when compared to the alternative solution: A TTP could deliberately transfer shared information and know-how to rival enterprises. For instance, confidential sensor data or blueprints could be leaked to competitors, which may then deduce quality

issues of the rival product. The service of the TTP could also be compromised by attackers, resulting e.g. in a violation of integrity so that the sharing parties receive inconsistent asset versions.

6 Evaluation

To evaluate our framework, Section 6.1 discusses the suitability of the framework in reference to the requirements. Finally, the results are discussed in Section 6.2.

6.1 Requirements fulfillment

To sum up, our approach fulfills the requirements **R1**-**R5**. The following paragraphs explain how each requirement was addressed in our solution architecture.

R1. Multi-party sharing. The main argument for using Distributed Ledgers is the involvement of multiple parties N who produce and consume data. Next to the ledger, our approach provides a client application for all parties that accesses the data on the ledger and the DHT. Therefore, our approach clearly fulfills **R1**.

R2. Data variety support. To enable the sharing of different data in various formats, our approach provides a central documentation and two storage options. The standardized asset description d_{spec} is included in the Distributed Ledger and serves as the basis of the DT within the specification contract. All other data of D_{desc} as well as the sensor data D_{sensor} are stored off-chain in the DHT. Moreover, each stored data element in DHT is registered in the central specification contract as a reference to the storage location of the data element. For instance, a sensor in the specification contract contains a reference to the storage position of its data stream in the DHT. Hence, **R2** is met.

R3. Data velocity support. Modern sensor data streams' frequency and volume exceed the performance characteristics of current Distributed Ledger frameworks. Since the data streams D_{sensor} do not describe main features of the DT (d_{spec}), they are stored off-chain in the DHT. This way, high throughput of D_{sensor} is supported, while the sharing latency is also kept low (seconds). The Distributed Ledger maintains verifiability by storing the hash reference to the data stream on the DHT in the specification contract. This ensures no loss in performance and data access through the DHT, supporting **R3**.

R4. Data integrity and confidentiality mechanisms. With respect to data integrity, the Distributed Ledger attaches every new data element ($trace()$) and prevents manipulation of the data by replicating it among all involved parties. A manipulation would result in a version mismatch or loss of consensus and could be detected easily ($audit()$). The second storage component (DHT) also supports integrity by storing the respective hash values to the data. A manipulation of DHT data would also be detected by a mismatch between the hashes in the nodes ($audit()$). However, there remains the problem of adding non-valid data, which is a common issue in the area of DLT. Here, we rely on the parties' interest in sharing valid data and on mechanisms ensuring quality of input data that the respective responsible party applies.

In terms of data confidentiality, our approach ensures that the data is read only by authenticated and authorized parties. Authentication is ensured through lifecycle party login to the client application (*authenticate*()). Access control concerning the party and the data elements is realized through an ACL and encryption for off-chain data and an authorization smart contract for on-chain data (*authorize*()). In concrete terms, the ACLs specify access rights on a per-document basis, while the smart contract stores authorization information for all involved parties. Therefore, different confidentiality levels can be realized.

To conclude, our approach provides data integrity and confidentiality mechanisms (**R4**) – reinforcing data security in DT data sharing.

R5. Read and write operations. Read and write operations are managed through the Client application. For *read*() operations, the Client application fetches the requested data from the DHT and the ledger and presents the data in a comprehensive way adjusted for the demanding party. In case of a *write*() operation, the Client application triggers the right procedure to alter the smart contract with a transaction and uploads additional asset-relevant data beyond specification to the DHT. Consequently, our approach also fulfills **R5**.

6.2 Discussion

Keeping the requirements *variety* **(R2)** and *velocity* **(R3)** in mind, the question arises why data *volume* is not considered a requirement. As literature is currently not at consensus regarding the relevance of the Big Data feature *volume* [15] for Digital Twins, we consider explicit support for data volume to be non-necessary. Nevertheless, our approach can deal with a fair amount of data.

It should be noted that our approach depends on multi-party participation. The more independent parties maintain the Distributed Ledger and DHT, the less vulnerable the data sharing is to manipulation. With regard to the access control capability, a decentralized identity management solution with a shared identity database could be an even more holistic, next-generation solution.

While we are aware that our approach currently lacks an implementation, we nevertheless believe that the use case shows suitability for practice. Future work will focus on implementing the framework. Here, challenges might include adjusting a DHT framework to support authorization and data feeds (although Swarm shows promise in this regard [7]), as well as selecting a suitable Distributed Ledger framework.

The Distributed Ledger and the concomitant smart contracts could also be handled in a different way. For instance, the AML could be transformed into classes and types in the smart contract, similar to the BPMN to Solidity transformation in [27]. However, the effort clearly outweighs the utility as AML is a very powerful standard allowing very complex descriptions. Moreover, not all of the hypothetically generated classes and functions might be needed. Plus, functions or classes might be newly added later on, which results in the need to re-create the smart contract as they are currently not represented in the smart contract. This clearly increases effort and downgrades utility.

Another issue is entailed by the possibility to directly alter variable values referring to an actual function in our current version of the ledger. For instance, consider a PLC device with various functions such as setting a conveyor belt's velocity (with an integer parameter). Without constraints, the changed velocity could exceed safety bounds. Safety threats like this one, be they malicious or accidental, need to be mitigated in a production system. Therefore, we suggest integrating safety and security rules as proposed in [6]. They could be integrated as part of the specification contract, with the Device agent checking conformance of program calls on synchronization.

With respect to the current problems hampering secure DT data sharing, our approach tackles the issues stated in Chapter 3 in the following ways:

- The usage of different tools that can be connected with our main data sharing approach is possible (*application of different tools*)
- Our approach is tailored for the integration of data in multiple formats and variety (*usage of various data formats*)
- An agreement only on the standard describing the asset (e.g. AML) is required to transform the main description of the asset into a smart contract, while other standardized or non-standardized data can still be shared (*missing standards*)
- The proposed shared collaborative data basis is distributed among all involved parties and the information flow is universal across the lifecycle phases (*broken information flow across lifecycle phases*)
- The Distributed Ledger registers the data and the involved party sharing the data, while mechanisms such as access control support confidentiality issues (*clarification of the ownership of information*)

To sum up, the major part of the identified issues in the literature referring to DT data sharing are diminished or solved by our approach.

7 Conclusion

DT data not only ties physical and virtual twin [23], it also enables integration of the whole asset lifecycle, which is essential for realizing the DT paradigm. Moreover, the exchange of asset-relevant data (DT data) is vital for achieving the effects of a feedback loop. Closing the feedback loop in turn favors the development of a circular economy.

However, maintaining data security becomes a major requirement when sharing DT data between multiple parties, especially as the parties do not necessarily trust each other. Our approach of applying DLT can clearly solve this issue and enable secure multi-party data sharing. It provides confidentiality through access control arranged by usage of a smart contract. Moreover, data integrity is implicitly supported through the immutability of the original data in the ledger.

To conclude, our approach fulfills the requirements **R1**-**R5** for secure DT data sharing. Nevertheless, there remain minor drawbacks that need to be addressed in future research (see Section 6.2). Our upcoming work will focus on implementing our theoretical concept to demonstrate its feasibility in practice.

References

1. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolić, M., Cocco, S.W., Yellick, J.: Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In: Proceedings of the Thirteenth EuroSys Conference. p. 30:1–30:15. EuroSys '18, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3190508.3190538
2. Banerjee, A., Dalal, R., Mittal, S., Joshi, K.P.: Generating Digital Twin models using Knowledge Graphs for Industrial Production Lines. In: Workshop on Industrial Knowledge Graphs. pp. 1–5. No. June (2017), http://ebiquity.umbc.edu/paper/html/id/779/Generating-Digital-Twin-models-using-Knowledge-Graphs-for-Industrial-Production-Lines
3. Baumgartner, R.J.: Nachhaltiges Produktmanagement durch die Kombination physischer und digitaler Produktlebenszyklen als Treiber für eine Kreislaufwirtschaft. In: Interdisziplinäre Perspektiven zur Zukunft der Wertschöpfung (2018). https://doi.org/10.1007/978-3-658-20265-1_26
4. Boschert, S., Heinrich, C., Rosen, R.: Next Generation Digital Twin. In: Proceedings of TMCE 2018. No. May (2018), https://www.researchgate.net/publication/325119950
5. Di Francesco Maesa, D., Mori, P., Ricci, L.: Blockchain based access control. In: IEEE Blockchain Conference 2018. pp. 1379–1386 (2018). https://doi.org/10.1007/978-3-319-59665-5_15
6. Eckhart, M., Ekelhart, A.: Towards Security-Aware Virtual Environments for Digital Twins. In: Proceedings of the 4th ACM Workshop on Cyber-Physical System Security - CPSS '18. pp. 61–72 (2018). https://doi.org/10.1145/3198458.3198464
7. Ethereum Swarm Contributors: Swarm 0.3 documentation (2019), https://readthedocs.org/projects/swarm-guide/downloads/pdf/latest/
8. Glaessgen, E., Stargel, D.: The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles. In: 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference (2012). https://doi.org/10.2514/6.2012-1818
9. Greengard, S.: Building a Better Iot (2017), https://cacm.acm.org/news/218924-building-a-better-iot/fulltext
10. ICS-CERT: Overview of cyber vulnerabilities. Tech. rep. (2017), https://ics-cert.us-cert.gov/content/overview-cyber-vulnerabilities
11. Kieselmann, O., Wacker, A., Schiele, G.: k-rAC - a Fine-Grained k-Resilient Access Control Scheme for Distributed Hash Tables. In: Proceedings of the 12th International Conference on Availability, Reliability and Security, ARES '17, Reggio Calabria, Italy. pp. 1–43. ACM, New York, NY, USA (8 2017). https://doi.org/10.1145/3098954.3103154
12. Litke, A., Anagnostopoulos, D., Varvarigou, T.: Blockchains for Supply Chain Management: Architectural Elements and Challenges Towards a Global Scale Deployment. Logistics **3**(1) (2019). https://doi.org/10.3390/logistics3010005
13. Malakuti, S., Grüner, S.: Architectural aspects of digital twins in IIoT systems. Proceedings of the 12th European Conference on Software Architecture Companion Proceedings - ECSA '18 pp. 1–2 (2018). https://doi.org/10.1145/3241403.3241417
14. Meroni, G., Plebani, P.: Combining Artifact-Driven Monitoring with Blockchain: Analysis and Solutions. In: Advanced Information Systems Engineering

Workshops. pp. 103–114. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-92898-2_8
15. Negri, E., Fumagalli, L., Macchi, M.: A Review of the Roles of Digital Twin in CPS-based Production Systems. Procedia Manufacturing **11**(June), 939–948 (2017). https://doi.org/10.1016/j.promfg.2017.07.198
16. Ovtcharova, J., Grethler, M.: Beyond the Digital Twin – Making Analytics come alive. visIT [Industrial IoT – Digital Twin] pp. 4–5 (2018), `https://www.iosb.fraunhofer.de/servlet/is/81714/`
17. Ríos, J., Hernández, J.C., Oliva, M., Mas, F.: Product avatar as digital counterpart of a physical individual product: Literature review and implications in an aircraft. In: Advances in Transdisciplinary Engineering (2015). https://doi.org/10.3233/978-1-61499-544-9-657
18. Rubio, J.E., Roman, R., Lopez, J.: Analysis of cybersecurity threats in industry 4.0: The case of intrusion detection. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (2018). https://doi.org/10.1007/978-3-319-99843-5_11
19. Sandhu, R.S., Samarati, P.: Access Control: Principles and Practice. IEEE Communications Magazine (1994). https://doi.org/10.1109/35.312842
20. Schroeder, G.N., Steinmetz, C., Pereira, C.E., Espindola, D.B.: Digital Twin Data Modeling with AutomationML and a Communication Methodology for Data Exchange. IFAC-PapersOnLine **49**(30), 12–17 (2016). https://doi.org/10.1016/j.ifacol.2016.11.115
21. Talkhestani, B.A., Jazdi, N., Schloegl, W., Weyrich, M.: Consistency check to synchronize the Digital Twin of manufacturing automation based on anchor points. In: Procedia CIRP (2018). https://doi.org/10.1016/j.procir.2018.03.166
22. Tankard, C.: The security issues of the Internet of Things. Computer Fraud & Security **2015**(9), 11–14 (9 2015). https://doi.org/10.1016/S1361-3723(15)30084-1
23. Tao, F., Cheng, J., Qi, Q., Zhang, M., Zhang, H., Sui, F.: Digital twin-driven product design, manufacturing and service with big data. International Journal of Advanced Manufacturing Technology **94**(9-12), 3563–3576 (2018). https://doi.org/10.1007/s00170-017-0233-1
24. Uhlemann, T.H., Lehmann, C., Steinhilper, R.: The Digital Twin: Realizing the Cyber-Physical Production System for Industry 4.0. In: Procedia CIRP (2017). https://doi.org/10.1016/j.procir.2016.11.152
25. Usländer, T.: Engineering of Digital Twins. Tech. rep., Fraunhofer IOSB (2018), `https://www.iosb.fraunhofer.de/servlet/is/81767/`
26. Voydock, V.L., Kent, S.T.: Security Mechanisms in High-Level Network Protocols. ACM Comput. Surv. (1983). https://doi.org/10.1145/356909.356913
27. Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., Mendling, J.: Untrusted business process monitoring and execution using blockchain. In: Lecture Notes in Computer Science (2016). https://doi.org/10.1007/978-3-319-45348-4_19
28. Wüst, K., Gervais, A.: Do you Need a Blockchain? In: 2018 Crypto Valley Conference on Blockchain Technology (CVCBT). pp. 45–54 (2018). https://doi.org/10.1109/CVCBT.2018.00011
29. Xu, X., Pautasso, C., Zhu, L., Gramoli, V., Ponomarev, A., Tran, A.B., Chen, S.: The blockchain as a software connector. In: Proceedings - 2016 13th Working IEEE/IFIP Conference on Software Architecture, WICSA 2016. pp. 182–191. IEEE (4 2016). https://doi.org/10.1109/WICSA.2016.21