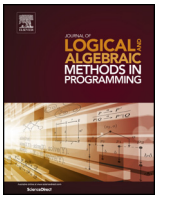
Journal of Logical and Algebraic Methods in Programming 134 (2023) 100886

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

journal homepage: www.elsevier.com/locate/jlamp



A formal model for blockchain-based consent management in data sharing

Neda Peyrone [a], Duangdao Wichadakul [a,b,*]

[a] Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok, 10330, Thailand
[b] Center of Excellence in Systems Biology, Faculty of Medicine, Chulalongkorn University, Bangkok, 10330, Thailand

ARTICLE INFO

Article history:
Received 8 December 2022
Received in revised form 1 May 2023
Accepted 23 May 2023
Available online 26 May 2023

Keywords:
GDPR
Data protection
Privacy by design
Consent management
Event-B
Smart contracts

ABSTRACT

Consent is one of six legal bases for personal data processing mentioned in the General Data Protection Regulation (GDPR). The GDPR is a privacy law giving European Union (EU) citizens authority over personal data. It enforces software systems to collect, analyze, and share only necessary information ('data minimization') following the specific purpose ('consent'). The GDPR defines consent as permission of individuals ('data subjects') to give organizations ('data controllers') processing their personal data. Without a data subject's consent, the data controller processes personal data unlawfully. Therefore, consent management is an essential component of a software system to build data subjects' trust and engagement. However, sharing data can lead to a potential loss of control over personal data, as data are across boundaries between software services. One of the significant risks is caused by a lack of developers' experience in data protection practices. Hence, in this paper, we propose to use blockchain technology to manage data subjects' informed consent for data sharing to build trust, transparency, and traceability to share data across software services. We formalized the semantics of smart contracts to extend the blockchain features to validate the consent authorization and manage the request-response interaction between the services. Furthermore, we used the Event-B method to describe the dynamic behavior of the proposed model and prove its correctness. Finally, we provided a mapping from the formal model to a smart contract class diagram and a prototype called SmartDataTrust implemented with solidity and Python REST API that developers can easily utilize.

1. Introduction

The GDPR [1] has heavily influenced businesses globally, which came into force on May 25, 2018. One of the challenges to software systems is how to share data in a way that protects the rights and interests of individuals [2–5]. Therefore, developers need to understand the data protection regime. However, developers find these regulations hard to interpret and implement into software engineering practices [6,7]. Furthermore, the lack of implementation guidelines on data protection causes data breaches (i.e., personal data is accidentally lost or compromised). Software systems that fail to meet GDPR requirements face suspension orders and heavy fines.

The GDPR affects software systems to handle personal data and guarantees the safeguarded data protection rights for EU citizens. It is a regulation in EU law that gives data subjects various rights to control their personal data. The entities

* Corresponding author at: Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok, 10330, Thailand.
E-mail addresses: peyrone.n@gmail.com (N. Peyrone), duangdao.w@chula.ac.th (D. Wichadakul).

https://doi.org/10.1016/j.jlamp.2023.100886
2352-2208/

involved in personal data processing include the data controller and data processor. The data controller is responsible for determining appropriate safeguards and ensuring that data subjects can exercise their rights. The data processor is responsible for implementing the task of personal data processing given by the data controller. The consent under Articles 6(1a) and 7 GDPR is divided into four elements of the following: 1) the data subject has free will to give his/her consent, 2) the purpose for which personal data is processed must be specific and clear, 3) the data controller must inform the data subject before data is gathered, and 4) the data subject gives unambiguous consent for the processing of his/her personal data. According to GDPR, consent is essential to ensure the freedom of data subjects to make decisions about their personal data. Hence, consent management will be a crucial component of a software system that handles permission (i.e., data subjects' consent) for personal data processing throughout its lifecycle.

Consent management provides the ability to manage data subject consent that covers GDPR or other protection regulations. To embed data protection as the core functionality of software systems, developers should incorporate data privacy by design. Privacy by Design (PbD) [8–10] is an approach that considers data protection upfront by embedding it into software systems. Using PbD increases privacy awareness and ensures compliance with GDPR requirements.

The GDPR provides guidelines, not on how to apply a privacy pattern [11]. On the other hand, PbD is a design methodology that includes data protection as the default setting for software systems. In addition, PbD is based on the seven Foundational Principles, which reveal guidelines for developers as follows: 1) it is essential to adopt a privacy-first approach to software design, 2) data protection must include in software systems by default, 3) a data protection mechanism must be an integral of software architecture, 4) the system must keep personal data confidential and accurate without reducing its ability, 5) the system must specify a retention period for the personal data collected and destroyed, 6) the system must provide privacy policies, and communication should be clear and transparent to individuals about data is being used, and 7) the system must protect personal data and respect individuals' privacy with a high level of security. In this paper, we thus propose a formal model for consent management in data sharing by embracing PbD.

Blockchain infrastructure provides the ability to manage data subjects' consent, which delivers integrity and immutability of given information [12–14]. A blockchain is a distributed ledger that uses a distributed network of nodes, and all nodes maintain a consistent copy of ledgers [15,16]. The distributed ledger is a special kind of database that offers a decentralized multi-version concurrency control and consensus mechanism (i.e., all nodes maintain the same distributed ledger). Nevertheless, cooperation between data privacy and blockchain technology benefits protecting data against manipulation.

In this paper, we propose to use the Event-B method [17] to model blockchain-based data sharing to comply with the requirements of PbD. Formal methods play a crucial role in data protection [18,19]. They use a mathematical approach to modeling and analyzing software requirements in the early phase of software development. To develop an Event-B model, we need to install the Rodin Platform [20] based on Eclipse IDE with instructions: http://www.event-b.org/install.html. The Rodin Platform provides an open-source tool that supports various plugins, such as a proof obligation generator, provers, and a model-checker (ProB). The advantage of using Event-B is its automatic prover, which provides practical support for model development and checking.

The proposed model explains the behavior of blockchain-based consent management in data sharing. It covered the main aspects of consent management (Article 4(11) GDPR), including managing data subjects' consent and determining authorized access for sharing personal data between services based on the given consent (Articles 5 & 20 GDPR), allowing data subjects to withdraw their consents and stop sharing data (Articles 17 & 19 GDPR), and allowing data subjects to renew their consents for continued use data sharing services (Article 6(1a) GDPR). Moreover, we implemented a Python REST API with smart contracts written in Solidity to enable consent management in data sharing, called SmartDataTrust. The platform has the complete development and integration tests performed on a local blockchain with Ganache (i.e., an Ethereum-like network emulator). The source code for local deployment is at https://github.com/cucpbioinfo/SmartDataTrust.

2. Related work

Several studies on managing consent and privacy-related personal data used blockchain technology. For example, Daudén-Esmel et al. [21] proposed a lightweight blockchain-based GDPR-compliant personal data management system. This study focused on a human-centric approach to managing personal data leveraged using smart contracts in blockchain. Moreover, the authors introduced conceptual design and system architecture for enabling individual data usage control under GDPR requirements. Thus, this architecture provides open-access immutable audit logs that record the interaction between data subjects and service providers under the data subjects' consent. In addition, it is designed to store personal data off-chain storage for increasing scalability and efficiency.

Similarly, Merlec et al. [22] proposed a smart-contract-based dynamic consent management system (SC-DCMS). This research aimed to study a human-centric approach to managing personal data and dynamic consent through smart contracts on a blockchain. By its design, data subjects or third-party organizations must first create dataset profiles that hold hashed indexes and use them to refer to individuals' data on off-chain, i.e., Secure Data Storage Network. Second, data profiles are published to the blockchain after peer data controllers approve. Moreover, SC-DCMS offers a smart contract code generator and predefined contract templates (i.e., defined in a JSON format for the XACML). The smart contract code generator translates consent and policies into the source code, indicating one consent agreement per smart contract. Within this proposed, smart contracts are designed to check the validity and expiration date of consent agreements and enable the audit logs that

track the activities of stakeholders (e.g., data subjects, data controllers, regulators, data processors) on consent agreements and personal data.

As the Internet of Things (IoT) grows, some IoT devices collect personal information, including health, behavioral, or biometric data. Rantos et al. [23] proposed the ADvoCATE approach, which adopted a human-centric approach to provide data subjects to manage their personal data in the IoT ecosystem regarding the GDPR requirements. In addition, ADvoCATE used blockchain technology to preserve data subjects' privacy interests by maintaining the integrity and versioning of data subjects' consent and provided an intelligence component to detect conflicts and offer recommendations on data subjects' policy.

Recently, data sharing has received growing attention in academic research and among practices. In particular, the topic of blockchain-based medical data sharing, and thus numerous studies have been published. For example, Azaria et al. [24] introduced MedRec to manage patients in electronic medical records (EMRs) across providers using smart contracts on the Ethereum blockchain. As for MedRec, smart contracts restrict access rights to such data in the patient's EMR. The system first maps participants' identification to their Ethereum address via Registrar Contract (RC) to enable the exchange of data between participants (i.e., patients and providers). Second, the system executes Patient-Provider Relationship Contract (PPR) to create a pairwise data exchange between patients and providers. The PPR defines data pointers that identify where data are collected and determines the permissions for third parties who desire to access data. Finally, the use of Summary Contract (SC) helps keep track of participants' engagement. Besides, MedRec uses mining reward techniques to encourage medical stakeholders to participate in the system as blockchain miners who validate transactions and blocks.

Similarly, Hu et al. [25] argued that the problem of data fragmentation creates issues for gathering a patient's data from heterogeneous service providers. The authors proposed CrowdMed-II, which improved from their previously proposed support for large-scale adoption called CrowdMed [26]. CrowdMed-II is a health data management framework based on the Ethereum blockchain to facilitate healthcare data sharing. The framework allows patients to control their health data by giving and revoking any permission they consented to. Utilizing blockchain in the framework builds transparency, auditability, and incentives and encourages patients to share their data to improve research outcomes. The CrowdMed-II is divided into three layers: 1) the data storage layer is an existing provider's database, 2) the central management layer comprises the central query manager and the blockchain, and 3) the user layer consists of four user roles: patients, data creators, data viewers, and data reviewers. The authors focused on designing two smart contract structures in this study, including Patient-Viewer Relationship (PVR)-Centric contract and Provider-Patient-Viewer Relationship (PPVR)-Centric contract structures. The PVR structure is similar to the PPR structure in MedRec [24]. The difference is that the PPR structure to retrieve a patient's medical records across providers needs multiple PPR contracts, while the PVR structure only requires a PVR contract. As for PPVR, it is adapted from the PPR and PVR structures. In addition, the authors added two more smart contracts: 1) the Provider Contract (PC) belongs to medical service providers and contains all patients' health data, and 2) the ReViewer Contract (RVC) has a similar function to PC, and data reviewers are a special kind of provider who reviews and comments on providers' data to improve its quality. However, data reviewers do not have their own database; thus, all the comments are stored in the health data sharing system. Furthermore, the system provides group-based access control to facilitate managed access rights instead of granting access per user.

In another study, Rouhani et al. proposed MediChain [27], implemented based on Hyperledger Fabric (HF) using discretionary access control. The HF is open-source with built-in features to allow access to only authorized peers, also known as permissioned (or private) blockchains. MediChain enables patients to manage and share their medical records with other patients, caregivers, and health practitioners. First, the patient requests the medical clinic to submit his/her medical assets into MediChain, and then the medical clinic notifies the caregiver. Second, the caregiver requests the patient access to his/her medical assets, and the patient decides whether to accept or decline the terms of access. If the patient agrees with the terms of access, the system creates a transaction on MediChain. After creating the transaction, the caregiver receives approval for accessing the patient's medical assets. Otherwise, if the patient declines the terms of access, no action is performed. Moreover, the patient's medical assets are encrypted and stored on off-chain storage servers, but only a hash of the asset's URI is stored in the blockchain.

There are commercial platforms that use blockchain to maintain patients' privacy. For example, Medicalchain [28] is an electronic health record platform based on a dual blockchain structure. The first structure uses access control lists of HF to restrict access to medical records and allows patients to control which portion of their data can be accessed. The second structure uses ERC20 tokens on Ethereum, including applications and services to support Medicalchain. However, tokens in Medicalchain are used for patients to pay doctors for telemedicine consultations. The platform enables patients to have complete control over their medical records. In addition, the platform uses symmetric key cryptography to encrypt patients' medical records to maintain privacy and confidentiality.

However, consent is an essential component in various domains. There are many studies focused on integrating consent management with blockchain technology. For example, Agarwal et al. introduced Consentio [29], implemented based on HF using smart contracts. Consentio allows data subjects to grant their consent for sharing the data with third parties and the ability to revoke their consent to remove third parties access. Data subjects must first assign roles to specify which third parties can perform on their data. Besides, the system enables audit logs to help data subjects monitor who requested access to their data. The world state of HF is a key-value type data store that holds the current state of the ledger. In this proposed design, the authors simplified key-value pairs of world states to improve the time complexity of reading and writing on three functionalities: assigning or revoking roles, granting or revoking consent, and requesting access. They

Table 1
Comparison of related works with our proposed model.

	[21]	[22]	[23]	[24]	[25]	[27]	[28]	[29]	[30]	[31]	Our work
GDPR compliant	✓	✓	✓	X	X	X	X	✓	✓	✓	✓
Formal method	X	X	X	X	X	X	X	X	X	X	✓
Data sharing among systems	X	X	X	✓	✓	X	X	X	X	X	✓
Consent management											
Access restriction based on the purpose or consent	✓	✓	✓	X	X	X	X	✓	✓	✓	✓
Manipulation	✓	✓	✓	X	X	X	X	✓	✓	✓	✓
Withdrawal	✓	✓	✓	X	X	X	X	✓	✓	✓	✓
Portability	X	X	X	X	X	X	X	X	X	X	✓
Renewal	X	X	X	X	X	X	X	X	X	X	✓
Auditability	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Limited the number of records in data selection	X	X	X	X	X	X	X	X	X	X	✓
Blockchain platform	Ethereum	Quorum	Ethereum	Ethereum	Ethereum	HF	HF/Ethereum	HF	HF	HF	Ethereum
Type of network	Public	Consortium	Public	Public	Public	Private	Private/Public	Private	Private	Private	Public
Use of off-chain storage	✓	✓	✓	✓	✓	✓	✓	✓	X	✓	✓
Implementation	Prototype	Prototype	Prototype	Prototype	Prototype	Prototype	Production	Prototype	Prototype	Prototype	Prototype
Performance Evaluation	X	✓	X	X	✓	X	X	✓	X	X	X

observed the time complexity of these functionalities associated with the following entities: Individual-oriented World State (IWS), Resource-oriented World State (RWS), and Role-oriented World State (RoWS). The studies revealed that the time complexity of request access transactions using RWS and RoWS was O(n) because there were key conflicts in consent modification transactions. Otherwise, the time complexity was O(1).

On the other hand, Agbo & Mahmoud [30] revealed that the risk causes a lack of transparency and audibility in health monitoring systems. They proposed designing a blockchain-based e-Health consent management framework based on HF, which helps data controllers ensure their systems comply with data protection regulations. The system enables patients to manage their health information and permission to either grant consent for partial or full access. Moreover, patients can freely revoke their consent at any time. The authors pointed out that an e-Health system with the support of consent management does not need to store a provider's storage. Based on the proposed design, the system should directly keep patients' records on the private blockchain because external applications cannot access the ledger.

According to Mamo et al. [31], dynamic consent in a biobank gives research partners (i.e., data subjects) more control over their biospecimens and personal data being used. The authors introduced a web portal based on blockchain technology called Dwarna. Dwarna was developed to support the dynamic consent process based on HF, which brings a great deal to genomic research efficiently and transparently. First, individuals willing to participate in genomic research must sign consent at the biobank before giving their biospecimen. Once a biospecimen has been received, the biobank generates a pseudonym for an individual as a research partner and then adds the individual's profile to Dwarna. Hence, a research partner can access the system and look at a list of ongoing research studies. The system allows research partners to choose and apply their consent to a particular study. Besides, they freely give or revoke their consent at any time, but in every request for consent alteration, the system will save transactions of consent change into the blockchain. Dwarna enables research partners to keep track of the use of their biospecimen through a history page.

The difference between our proposed model and related works is that other studies focused on conceptual and architectural frameworks, while our study focused on formalizing the model in aspects of logic that help to understand its behavioral specifications. Moreover, our proposed model aimed to share data among different platforms without copying any customer data into secure storage servers (Table 1) because each platform has its own customer data to maintain. It will reduce both IT costs and operational burdens. In our model, smart contracts are reusable. Once smart contracts are deployed, platforms that exchange data only focus on implementing a REST API for consuming an API to manage consent on the blockchain and handling callback URLs made by the blockchain. Adopting our model will reduce developers' work and keep the system design as simple as possible.

3. Background

We reviewed data-sharing issues from the view of system design to build a GDPR-aware system model on blockchain related to PbD (Table 2). The critical roles under GDPR included 1) data subject, 2) data controller, and 3) data processor.

Table 2
Data sharing-related issues as requirements for blockchain-based consent management.

Topic	Issue	Requirement
Rules of data sharing upon a particular purpose	The challenge of consent associated with sharing personal data is to manage consent and personal data effectively and transparently [2–5]. The data subject can control and provide his/her consent over personal data being shared [2,3,5,32].	**RQ1:** The system shall determine the consent management functionality based on decentralized security, which enables an immutable audit log and transparent data-sharing over a network.
Access restriction based on the purpose or consent	The data subject has the right to give his/her consent to transmit personal data among data controllers [2,3].	**RQ2:** The system shall allow data controllers to request and disclose individuals' data only if the data subject has provided his/her consent.
Limited number of records in data selection	The sharing of unnecessary personal data puts the confidentiality and privacy of individuals at risk [33,34]. The data controller is restricted to sharing only the minimum amount of personal data necessary [3,35,36].	**RQ3:** The system shall determine that one request-response interaction is only provided for one individual's data and it will be disclosed in accordance with predefined data fields in this given consent.
Consent withdrawal	The data subject has the right to revoke consent to discontinue sharing personal data as he/she wishes [2,12].	**RQ4:** The system shall provide a mechanism for consent revocation in which the data subject can revoke consent at any time.
Consent renewal	The data controller may request the data subject an extension of the retention period for continuing to share his/her personal data [2,12].	**RQ5:** The system shall provide a mechanism for consent renewal in which the data subject can renew consent.
Auditability	The sharing of personal data among data controllers should be documented at each transmission step in immutable and transparent data storage [2,3,34]. The data subject shall have access to audit log activities for tracking at each transmission step based on the consent that he/she provided [2,3,32].	**RQ6:** The system shall provide a mechanism for audit logging to document a request-response interaction between participant data controllers on every disclosure.
Personal data masking	The risk of direct personal identification in data sharing may cause the recognition of individual persons [34,37–39].	**RQ7:** The system shall provide a mechanism for pseudonymized data to reduce the risk of identifying individual persons through data sharing.
Secure distributed data storage	The sharing of personal data among data controllers should not duplicate any individuals' data in secure distributed data storages, reducing IT costs and operational burdens.	**RQ8:** The system shall enable async callback to manage the request-response interaction with dynamic configuration endpoint callback URLs for eliminating the use of secure distributed data storage.

Article 4(1) GDPR defined 'personal data' as direct or indirect information that recognizes an individual. A data subject has the right to control the processing of his/her personal data. The data controller is the organization or person who determines the purpose and means of personal data processing, as described in Article 4(7). The data processor is the organization or person who conducts personal data following the data controller's direction, as described in Article 4(8). Therefore, we defined the data sharing state machine (DSSM) upon requirements in Table 2 that covered blockchain-enabled consent management in data sharing and created a mapping of GDPR articles relevant to DSSM in Table 3. This state machine aims to help developers address GDPR requirements in software engineering practices.

To define a set of states and transitions in DSSM, we determined the logic within consent management functionality comprises the following fundamental features: 1) the consent authorization feature is used to restrict access to share personal data based on the given consent (Articles 5 & 20 GDPR), 2) the consent withdrawal feature is used to revoke permission to share personal data (Articles 17 & 19 GDPR), and 3) the consent renewal feature is used to keep data sharing functionality available (Article 6(1a) GDPR). The consent authorization feature is essential in data-sharing processing activities to check whether consent is expired or withdrawn based on the data subject's consent. If consent is expired or removed, data transfer is not permitted. Otherwise, the system can proceed with data-sharing activities, i.e., transfer data to another service. This feature involves processing and portability activities following Articles 5 and 20 GDPR. Article 5 lays down the rules applying to personal data processing that respects six data protection principles as follows: 1) it requires personal data to be collected and used for legitimate purposes only ('lawfulness, fairness, and transparency'), 2) the purpose must be limited and precise before beginning any data processing activities ('purpose limitation'), 3) personal data should include a minimum amount of data and be limited to a specific purpose ('data minimization'), 4) it entails that personal data should be complete and kept-up-to-date ('accuracy'), 5) the data controller must ensure not to maintain personal data longer than necessary ('storage limitation'), and 6) the data controller must ensure the confidentiality and integrity of personal data in its process ('integrity and confidentiality'). As for Article 20, the right to data portability gives data subjects the right to transfer personal data from one service to another in a machine-readable format.

Furthermore, the consent withdrawal feature is involved in revoking consent and deleting personal data following Articles 7(3), 17, and 19 of GDPR. Article 7(3) describes that the data subject who gave consent has the right to withdraw the consent at any time. In Articles 17 and 19, the data subject has the right to ask for his/her personal data to be erased, also known as the right to be forgotten. Within our proposed model, the right to erasure has been automatically included in the system [20] because it does not obtain personal data on the blockchain or off-chain storage servers.

Table 3
The proposed model and GDPR articles it covered.

	DSSM			Class Diagram			GDPR article
	Event	Set/Constant	Local/State variable	Operation	Class	Attribute	
RQ1	AddConsent	CONSENTS	consents	addConsent	Consent:struct, ConsentContract	consentCode, consentDetail, consentVersion, dataRetention, requesterId, requesterUrl	Article 4(1), Article 4(4), Article 4(7), Article 5(1b), Article 24, Article 28, Article 37
		CONSENTS, FIELDS	dataFields	addDataField	DataField:struct, DataFieldContract	consentCode, consentVersion, fieldName	
	AddDataSubjectConsent	PARTICIPANTS, DATA_SUBJECTS, CONSENTS, BOOL	dataSubjectConsents	addDataSubjectConsent	DataSubjectConsent:struct, DataSubjectConsentContract	responderId, responderUrl, pseudonym, consentCode, consentVersion	
RQ2		PARTICIPANTS, DATA_SUBJECTS, CONSENTS, BOOL	consentExpired:bool, dataSubjectConsents	isConsentValid	DataSubjectConsent:struct, DataSubjectConsentContract	responderId, pseudonym, consentCode, consentVersion, dataRetention, createTimestamp	Article 5(1a), Article 5(1c), Article 5(1d), Article 6(1a)
RQ3	SubmitRequest	REQUESTS, PARTICIPANTS, DATA_SUBJECTS, CONSENTS	dataAccessRequests	submitRequest	DataAccessRequest:struct, DataAccessRequestContract	requestExists, requestId, pseudonym, consentCode, consentVersion	Article 5(1e), Article 5(1f), Article 6(1a), Article 20
	SubmitResponse	RESPONSES, REQUESTS	dataAccessResponses	submitResponse	DataAccessResponse:struct, DataAccessResponseContract	responseExists, responseId, requestId, pseudonym, consentCode, consentVersion	
RQ4	RevokeConsent	PARTICIPANTS, DATA_SUBJECTS, CONSENTS, BOOL	dataSubjectConsents	revokeConsent	DataSubjectConsent:struct, DataSubjectConsentContract	pseudonym, responderId, consentCode, consentVersion, withdrawnTimestamp	Article 7(3), Article 17, Article 19
RQ5	RenewConsent	PARTICIPANTS, DATA_SUBJECTS, CONSENTS, BOOL	dataSubjectConsents	renewConsent	DataSubjectConsent:struct, DataSubjectConsentContract	pseudonym, responderId, consentCode, consentVersion, createTimestamp	Article 6(1a)
RQ6				LogAddedConsent, LogInactivatedConsent	ConsentContract		
				LogAddedDataField	DataFieldContract		
				LogAddedDataSubjectConsent, LogFiredRequesterCallback, LogReturnedRequesterCallback, LogRevokedConsent, LogRenewedConsent	DataSubjectConsentContract		
				LogSubmittedRequest, LogFiredResponderCallback, LogReturnedResponderCallback	DataAccessRequestContract		
				LogSubmittedResponse, LogFiredDataTransferCallback, LogReturnedDataTransferCallback	DataAccessResponseContract		
RQ7					DataSubjectConsent:struct	pseudonym	Article 4(5)
RQ8	CallbackRequester	balanceOf(this), PARTICIPANTS, DATA_SUBJECTS, CONSENTS, BOOL	oraclizeFee:number, dataSubjectConsents, callbackRequesterStates	callbackRequester	DataSubjectConsent:struct, DataSubjectConsentContract	requestId, pseudonym, consentCode, consentVersion, requesterUrl	
	CallbackResponder	balanceOf(this), REQUESTS, PARTICIPANTS, DATA_SUBJECTS, CONSENTS	oraclizeFee:number, dataAccessRequests, callbackResponderStates	callbackResponder	DataAccessRequest:struct, DataAccessRequestContract	requestId, pseudonym, consentCode, consentVersion, responderUrl	
	CallbackDataTransfer	balanceOf(this), RESPONSES, REQUESTS	oraclizeFee:number, dataAccessResponses, callbackDataTransferStates	callbackDataTransfer	DataAccessResponse:struct, DataAccessResponseContract	responseId, responderUrl, transferUrl	

Finally, the consent renewal feature is involved in renewing consent effects within Article 6(1a) GDPR. The data controller or data processor may offer a data subject to extend the retention period to continue using data-sharing services. If the data subject accepts the renewal retention period, the data controller or data processor allows continued processing of his/her personal data.

Blockchain technology enables potential data protection, for example, creating a secure and permanent record of transactions among untrusted parties, eliminating the need for centralized entities, and using cryptographic hash algorithms and distributed peer-to-peer networks to verify data integrity [12–14]. Blockchain is an ordered list of transactions at its core, called a block [14,40,41]. Once sufficient participants confirm the transaction, it is permanently added to the list of blocks, and each block is linked and secured using cryptography. The key components of a block are as follows: 1) the data could be a string or list of transactions, 2) the hash is a unique identifier created by a nonce, and the nonce is a random 32-bit whole number, 3) the previous hash is the hash value of the previous block, and 4) the metadata is information that describes the data, e.g., block number, timestamp. Moreover, blockchain uses consensus mechanisms or algorithms to ensure fault tolerance and security by allowing genuine participants to add new transactions. These consensus mechanisms assign participants to work on tasks or activities to keep blockchain infrastructure operational by dedicating resources [41–43], e.g., money and energy power. Recently, there have been two primary consensus mechanisms used by most cryptocurrencies. First, Proof of Work (PoW) is secured and verified transactions and new tokens by participants to compete to be the first to solve math puzzles, which requires massive computing power. Another, Proof of Stake (PoS), is similar to PoW; participants must purchase the cryptocurrency and hold it to get selected to form a block and earn rewards.

Therefore, the difference between PoW and PoS is that PoW requires participants to consume electricity, while PoS does not require participants to consume electricity on duplicate tasks or activities (i.e., solving the same math puzzles). Ethereum is an open-source blockchain with a PoS consensus mechanism and smart contract functionality. The use of Ethereum in data sharing here aims to record the request-response interactions between the requesters who wish to access the personal data and the responders who maintain the personal data. Besides, we implemented smart contracts to extend the blockchain features to validate the consent authorization and introduce the callback functions to manage the request-response interaction between requesters and responders.

In this paper, we used an Event-B method for refinement and verification. According to Khan et al. [42], formal verification is essential to evaluate smart contracts behavior, which guides in finding bugs and errors to ensure its correctness on given properties within all conditions. Moreover, several studies demonstrated that Event-B is suited for formalizing a smart contract [44,45]. An Event-B is a modeling method used to formalize systems in mathematical terms [46], divided into two parts: 1) the context, a static part of a model, which contains carrier sets *s*, constants *c*, and axioms *A(s, c)*, and 2) the machine, a dynamic part used to provide behavioral properties of the model, which contains the state variables *v*, invariants *I(s, c, v)*, and events. In addition, the Event-B model is required to preserve the consistency proof obligations (POs) for a set of events produced by each invariant rule.

The POs ensure that each event in the machine maintains the preserved property. The guards are used to specify preconditions for executing an event. For each event of a machine consisting of one or more guards *G(s, c, v)*, when guards are valid, the state variables v will be modified by actions *S(s, c, v)*, as shown in Equation (1).

$$when\ \ G(s,c,v)\ \ then\ \ v\ :|\ \ S(s,c,v)\ \ end \tag{1}$$

The POs are guaranteed that each event preserves invariants when a state changes, as shown in Equation (2).

$$\begin{array}{l} I(s,c,v) \\ G(s,c,v) \\ A(s,c) \\ \vdash \\ I(S(s,c,v)) \end{array} \tag{2}$$

Event-B is constructed based on a refinement process. The refinement process is essential for modeling a complex system by incrementing a small portion of features. Each refinement step gradually adds new events into the model to prove its correctness against the previous actions. It makes an Event-B model easier to verify its POs.

To begin with Event-B, we summarize some set-theoretical notations used on the proposed state machine. We denote set predicates by P and Q, set expressions by S, T, and E, single variables by x and y, a list of variables by z, and the relation by r, r_1, and r_2. The set-theoretical notations are as follows: 1) the predicate logic, e.g., conjunction ($P \wedge Q$), disjunction ($P \vee Q$), and existential quantification ($(\exists z \cdot P) \wedge Q$), 2) the pre-defined sets, e.g., booleans (BOOL), i.e., TRUE or FALSE, and empty set ($\varnothing$), 3) the set operators, e.g., membership ($E \in S$), union ($S \cup T$), intersection ($S \cap T$), powerset ($\mathbb{P}(S)$), a subset ($S \subseteq T$), not a subset ($S \nsubseteq T$), ordered pairs ($x \mapsto y$), set difference ($S \setminus T$), cartesian product ($S \times T$), and 4) the relations defining the relationship between sets, e.g., relations ($S \leftrightarrow T$), domain (dom(r)), range (ran(r)), partial functions ($S \nrightarrow T$), partial injections ($S \rightarrowtail T$), domain restriction ($S \lhd T$), domain subtraction ($S \hookleftarrow T$), range restriction ($S \rhd T$), range subtraction ($S \rhd T$), relational image (r[S]), and overriding ($r_1 \lhd\!\!+ r_2$). More detailed information about Event-B notation is available online at [47].

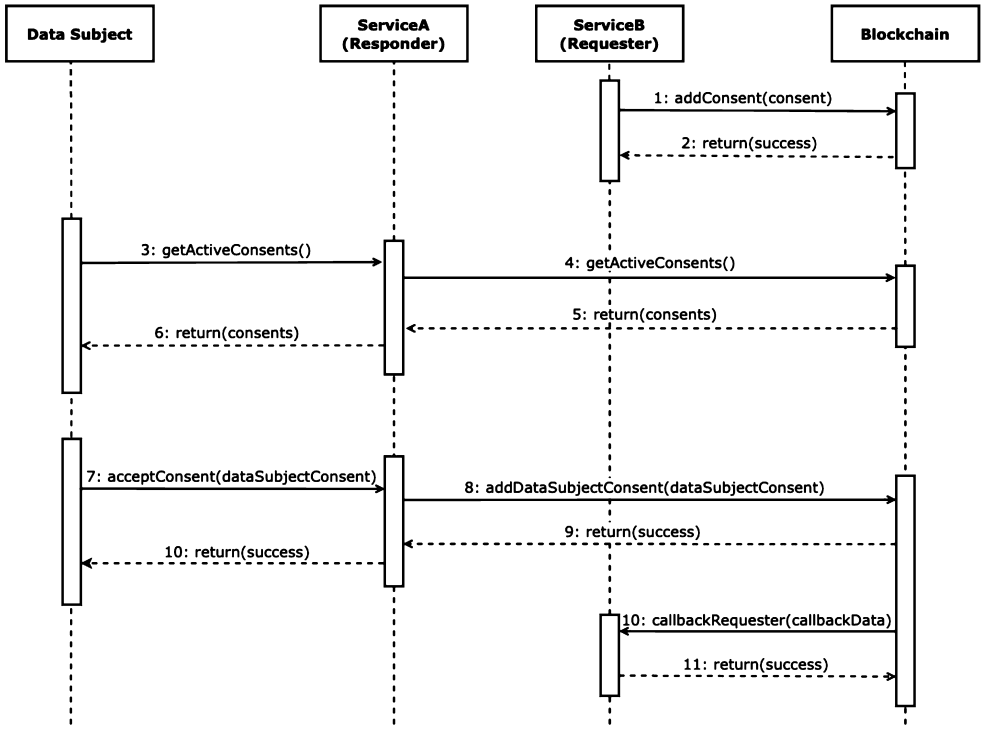


Fig. 1. Data sharing sequence diagram illustrating the request-response interaction between ServiceA (responder) and Service B (requester). First, ServiceB adds a new consent into the blockchain. Second, the data subject retrieves all available consents, then selects and accepts the ServiceB's consent through ServiceA. Third, ServiceA adds a new data subject's consent into the blockchain. The blockchain then creates a callback to the ServiceB.

4. Consent management in data sharing state machine

This research proposes a formal model which embeds data protection into software development upon the GDPR. Based on Article 4(11) GDPR, for the consent to be valid, the data subject voluntarily agrees to enable either a data controller or a data processor to process his/her personal data for a specific purpose. We considered consent management essential for promoting privacy awareness in the system design [48,49]. Furthermore, it indicates that the system cannot process or share personal data without the data subject's consent. In this paper, we built a state machine to depict the dynamic behavior of a requester sending requests to access personal data on the blockchain relevant to the relationships of a data subject's consent, a requester, a responder, and a smart contract's balance. We followed PbD concepts and GDPR guidelines presented in Table 3 and provided the example of request-response interaction through the data-sharing sequence diagram (Figs. 1 and 2) and the DSSM (Fig. 3) that covers the main aspects of blockchain-based consent management in data sharing. We utilize the blockchain to obtain records of all request-response interactions without storing personal data. Moreover, the requester and responder communicate through blockchain, which is strictly forbidden to communicate directly with each other. The interactions between the requester and the responder begin with the requester requesting to access personal data through smart contracts (i.e., providing consent management) live on a blockchain. Then smart contracts automatically check if the data subject has authorized access to their personal data. If the request is approved, the blockchain makes a callback to trigger the responder. Finally, the responder sends the response back to the blockchain and transmits personal data to the requester through an off-chain channel (i.e., the channel allowing transactions to occur outside the blockchain).

Based on DSSM, requesters first add their new consent into the blockchain. Second, the data subject accesses the front-end of his/her data provider, a responder, and retrieves from the blockchain all available consents required by requesters offering new products or services. The data subject must accept before using its products or services. Third, after the data subject agrees with a requester's consent, the responder sends back the data subject's acceptance status into the blockchain. Fourth, when the new data subject's consent has been stored on the blockchain, the blockchain makes a callback to trigger the requester, which can prepare a request for accessing personal data. Fifth, when the request has been stored on the blockchain, the blockchain makes a callback to trigger the responder, which can respond to access the personal data within the retention period. Sixth, when the response has been stored on the blockchain, the blockchain makes a callback to trigger the responder, which can transfer personal data directly to the requester via an off-chain channel. One request will get only one response in our model, as tracking all requests and responses on the blockchain is easier.

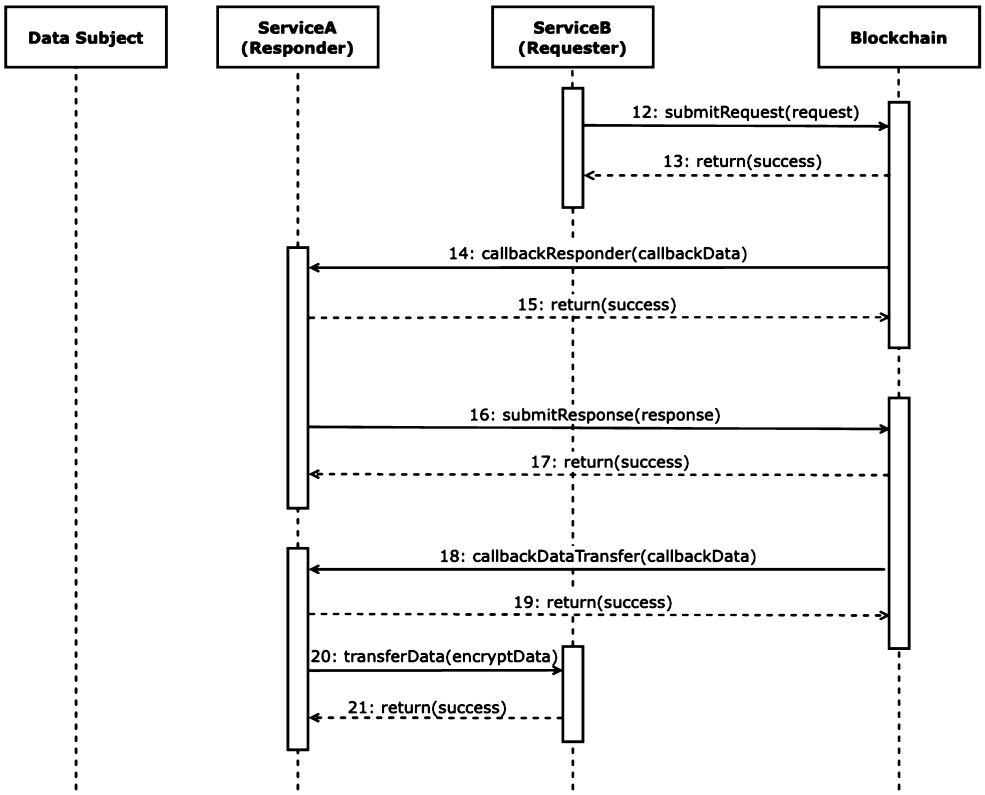


Fig. 2. Data sharing sequence diagram continued from the previous diagram (Fig. 1), which illustrates the request-response interaction between ServiceA and ServiceB. After ServiceB receives a callback, it prepares a request for accessing personal data and sends it to the blockchain. The blockchain then creates a callback to ServiceA in which ServiceA responds to a requested request within the retention period via the blockchain. Finally, the blockchain invokes a callback to ServiceA again. In doing so, ServiceA starts to transmit personal data to ServiceB directly.

5. Formal development in Event-B

To build the data sharing model, first, we created the data sharing context (DSCX) to define carrier sets, and constants associated with blockchain as follows: 1) CONSENTS is a set of personal data sharing agreements (e.g., ConsentA, ConsentB) between the services sending and receiving data, 2) FIELDS is a set of data fields (e.g., Name, BirthDate, BirthDefects) that identifies individuals, 3) DATA_SUBJECTS is a set of data subjects (e.g., DataSubject1), 4) PARTICIPANTS is a set of services (e.g., ServiceA, ServiceB) that require data sharing based on blockchain technology, 5) ADDRESSES is a set of contract addresses to interact with deployed smart contracts, 6) the constant *this* is a member of ADDRESSES, which refers to the contract address itself, 7) the constant *initialBalance* is a natural number representing the initial balance of contract address *this*, 8) REQUESTS is a set of data requests, the requesting services (requesters) create requests (e.g., Request1) to the responding services (responders) for accessing personal data, and 9) RESPONSES is a set of data responses, the responders check whether the requests have authorized access to personal data and return the responses (e.g., Response1) back to the requesters. Second, we created DSSM and referred to DSCX; the state machine can then directly access the defined global static variables. State machine naturally encapsulates states and behaviors related to variables, invariants, and transitions. In the Event-B model, variables represent the states of the system, and invariants, e.g., inv1, inv2, represent the preserved properties of the states. A transition represents the changing from one state to another according to an event. Every event comprises guards as preconditions and actions for variable modification, which are labeled, e.g., grd1, grd2, and act1, act2, respectively. A transition will take place only if it satisfies all invariants and guards.

5.1. Data sharing state machine (DSSM)

The DSSM (Fig. 3) was modeled and formally proved for blockchain-based data sharing. It depicts the dynamic behavior of a requester sending requests to access personal data on the blockchain that provides consent-based access control. If the request is authorized, the responder will send the response back to the blockchain and transmit personal data to the requester through an off-chain channel. For this state machine, we defined the preserved invariants as follows:

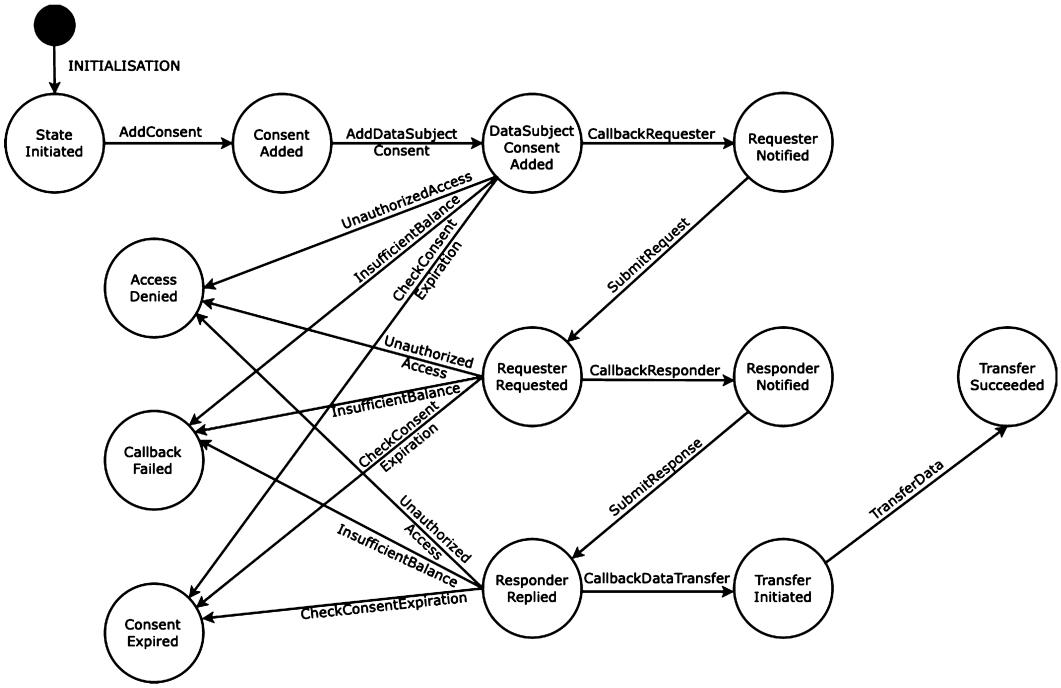


Fig. 3. Data Sharing State Machine (DSSM) illustrating the transition states and events used to share personal data between requesters and responders through blockchain.

5.1.1. Invariants in DSSM

inv1 : $consents \in \mathbb{P}(CONSENTS)$

inv2 : $dataFields \in CONSENTS \nrightarrow \mathbb{P}\,1(FIELDS)$

inv3 : $dataSubjectConsents \in PARTICIPANTS \times DATA\_SUBJECTS \times CONSENTS \nrightarrow BOOL$

inv4 : $addresses \subseteq ADDRESSES$

inv5 : $balanceOf \in addresses \rightarrow \mathbb{N}$

inv6 : $callbackRequesterStates \in \mathbb{P}(PARTICIPANTS \times DATA\_SUBJECTS \times CONSENTS)$

inv7 : $dataAccessRequests \in REQUESTS \nrightarrow PARTICIPANTS \times DATA\_SUBJECTS \times CONSENTS$

inv8 : $callbackResponderStates \in \mathbb{P}(REQUESTS)$

inv9 : $dataAccessResponses \in RESPONSES \rightarrowtail REQUESTS$

inv10 : $callbackDataTransferStates \in \mathbb{P}(RESPONSES)$

inv11 : $encryptedData \in RESPONSES \nrightarrow \mathbb{P}(DATA\_SUBJECTS \times FIELDS)$

inv12 : $dataTransferStates \in RESPONSES \nrightarrow BOOL$

The variable *consents* contains a set of CONSENTS, which holds all consents offered by the requesters. According to PbD, the requester must demonstrate that data subjects agreed to process their personal data for a specific purpose on the defined data fields. Hence, we declared the variable *dataFields* as a set of ordered pairs (consent $\mapsto$ dataField) where consent $\in$ CONSENTS and dataField $\in \mathbb{P}\,1$(FIELDS). The *dataFields* specifies that consent can have one or more data fields. The *dataSubjectConsents* defines (pdc $\mapsto$ active) as a set of ordered pairs, where pdc $\in$ PARTICIPANTS $\times$ DATA_SUBJECTS $\times$ CONSENTS, and active $\in$ BOOL (i.e., TRUE or FALSE). The variable *dataSubjectConsents* represents a record of the data subject's consent that allows a requester to process his/her personal data under the purpose of the consent. The *addresses* is a subset of the ADDRESSES set where each represents a unique smart contract address on the blockchain. We defined the *balanceOf* (address $\mapsto$ balance) where address $\in$ addresses and *balance* is a natural number, keeping track of the contract address balance. The variable *callbackRequesterStates* contains a set of ordered triples (responder $\mapsto$ dataSubject $\mapsto$ consent) where responder $\in$ PARTICIPANTS, dataSubject $\in$ DATA_SUBJECTS, consent $\in$ CONSENTS to track which requesters have been successfully invoked after the data subjects have given their consents for their data processing. The variable

AddConsent $\widehat{=}$

Any $consent, dataField$ **Where**

$grd1 : consent \in CONSENTS \land consent \notin consents$

$grd2 : dataField \in \mathbb{P}\,1(FIELDS)$

$grd3 : dataFields \lhd\!\!+ \{consent \mapsto dataField\} \in CONSENTS \nrightarrow \mathbb{P}\,1(FIELDS)$

Then

$act1 : consents := consents \cup \{consent\}$

$ac21 : dataFields(consent) := dataField$

End

Listing 1: The AddConsent event.

AddDataSubjectConsent $\widehat{=}$

Any $responder, dataSubject, consent$ **Where**

$grd1 : responder \in PARTICIPANTS$

$grd2 : dataSubject \in DATA\_SUBJECTS$

$grd3 : consent \in consents \land consent \in \mathrm{dom}(dataFields)$

$grd4 : responder \mapsto dataSubject \mapsto consent \notin \mathrm{dom}(dataSubjectConsents)$

$grd5 : dataSubjectConsents \lhd\!\!+ \{responder \mapsto dataSubject \mapsto consent \mapsto TRUE\} \in$

$(PARTICIPANTS \times DATA\_SUBJECTS \times CONSENTS) \nrightarrow BOOL$

Then

$act1 : dataSubjectConsents(responder \mapsto dataSubject \mapsto consent) := TRUE$

End

Listing 2: The AddDataSubjectConsent event.

dataAccessRequests is a set of ordered pairs (request $\mapsto$ dataSubjectConsent) where request $\in$ REQUESTS, and dataSubjectConsent $\in$ dom(dataSubjectConsents), represents a record of data request of a requester to the blockchain after receiving a callback of data subject's permission. The *callbackResponderStates* contains a set of REQUESTS to track which responders have been successfully invoked after the requesters have initiated their requests. Hence, we declared the *dataAccessResponses* that holds the one-to-one relationship between RESPONSES and REQUESTS. This mapping allows transferring data between the responder and requester. The variable *callbackDataTransferStates* contains a set of RESPONSES to track which responders have been successfully invoked for starting an off-chain data transfer after accepting the requests. We stored the encrypted data in variable *encryptedData*, a set of ordered pairs (response $\mapsto$ personalData) where response $\in$ RESPONSES, and personalData $\in$ DATA_SUBJECTS $\times$ FIELDS. Furthermore, we defined a variable *dataTransferStates* to hold the status of successful data transfer as a set of ordered pairs (response $\mapsto$ success) where response $\in$ RESPONSES, and success $\in$ BOOL (i.e., TRUE or FALSE).

5.1.2. Events in DSSM

The DSSM state machine is executed starting from the INITIALISATION event, then all variables of DSSM are initialized. Listing 1 shows the formal model of how a new requester's consent is added to the blockchain. The guards are defined with three preconditions. First, the guard grd1 ensures that the *consent* does not exist in the variable *consents*. Second, the guard grd2 ensures that any *dataField* is a member of $\mathbb{P}\,1$(FIELDS). Third, the guard grd3 ensures that adding an ordered pair (consent $\mapsto$ dataField) into variable *dataFields* must satisfy the invariant inv2. Whenever all guards are valid, action act1 adds the consent to the *consents*, and action act2 adds an ordered pair (consent $\mapsto$ dataField) to the *dataFields*.

Listing 2 shows how to formally model the adding of a new data subject's consent. The guards are defined with five preconditions. First, the guard grd1 ensures that the *responder* is a member of PARTICIPANTS. Second, the guard grd2 ensures that the *dataSubject* is a member of DATA_SUBJECTS. Third, the guard grd3 ensures that the *consent* is a member of the variable *consents* and within the domain *dataFields*. Fourth, the guard grd4 ensures that a new ordered triple (responder $\mapsto$ dataSubject $\mapsto$ consent) does not exist in the domain *dataSubjectConsents*, which means no active data subject's consent is already granted for the requester on the blockchain. Fifth, the guard grd5 ensures that when adding TRUE to the *data-*

CallbackRequester $\widehat{=}$

Any $oraclizeFee, dataSubjectConsent$ **Where**

$grd1 : this \in dom(balanceOf) \land oraclizeFee \in \mathbb{N} \land oraclizeFee \leq balanceOf(this)$

$grd2 : balanceOf \lhd\!\!+ \{this \mapsto balanceOf(this) - oraclizeFee\} \in addresses \rightarrow \mathbb{N}$

$grd3 : dataSubjectConsent \in \text{dom}(dataSubjectConsents) \land dataSubjectConsent \notin callbackRequesterStates \land$
$dataSubjectConsents(dataSubjectConsent) = TRUE$

Then

$act1 : balanceOf := balanceOf \lhd\!\!+ \{this \mapsto balanceOf(this) - oraclizeFee\}$

$act2 : callbackRequesterStates := callbackRequesterStates \cup \{dataSubjectConsent\}$

End

Listing 3: The CallbackRequester event.

SubmitRequest $\widehat{=}$

Any $consentExpired, dataSubjectConsent, request$ **Where**

$grd1 : consentExpired \in BOOL \land consentExpired = FALSE$

$grd2 : dataSubjectConsent \in dom(dataSubjectConsents) \land dataSubjectConsents(dataSubjectConsent) = TRUE$

$grd3 : dataSubjectConsent \in callbackRequesterStates$

$grd4 : request \in REQUESTS \land request \notin dom(dataAccessRequests)$

$grd5 : dataAccessRequests \lhd\!\!+ \{request \mapsto dataSubjectConsent\} \in$
$REQUESTS \nrightarrow PARTICIPANTS \times DATA\_SUBJECTS \times CONSENTS$

Then

$act1 : dataAccessRequests(request) := dataSubjectConsent$

End

Listing 4: The SubmitRequest event.

SubjectConsents(responder $\mapsto$ *dataSubject* $\mapsto$ *consent)*, the invariant inv3 must be satisfied. Finally, whenever all of the guards are valid, the action act1 adds TRUE to the *dataSubjectConsents(responder* $\mapsto$ *dataSubject* $\mapsto$ *consent)*.

Listing 3 shows how we formally model the handling of the request-response mechanism on the blockchain. After adding a new data subject's consent, the blockchain creates a callback to the requester (Listing 3) via an outside API call. When the requester receives a callback, it will start preparing a request for accessing personal data. The guards are defined with three preconditions. First, the guard grd1 ensures that the constant *this* is a member of the domain *balanceOf*, the *oraclizeFee* is the charge for sending a payload to an API call outside the blockchain, which is a member of a set of natural numbers, and the *oraclizeFee* must be less than or equal to *balanceOf(this)*. Second, the guard grd2 ensures that the decreased *balanceOf(this)* with the *oraclizeFee* must satisfy the invariant inv5. Third, the guard grd3 ensures that the *dataSubjectConsent* is a member of the domain *dataSubjectConsents*, *dataSubjectConsent* does not exist in the *callbackRequesterStates*, and the active status of the *dataSubjectConsents(dataSubjectConsent)* is TRUE. Whenever all guards are valid, action act1 charges *oraclizeFee* from the *balanceOf(this)*, and action act2 adds the *dataSubjectConsent* to the *callbackRequesterStates*.

The SubmitRequest event (Listing 4) allows a requester to create a request for accessing personal data. In this event, we defined a set of constraints to restrict the request access: 1) the consent has not expired, 2) the consent has not been withdrawn, and 3) the request ID has not been submitted. These constraints were then described as five guards of the event. First, the guard grd1 ensures that the *consentExpired* is a member of the boolean and *consentExpired* is FALSE. Second, the guard grd2 ensures that the *dataSubjectConsent* is a member of the *dataSubjectConsents*, and the range of *dataSubjectConsents(dataSubjectConsent)* is TRUE. Third, the guard grd3 ensures that the *dataSubjectConsent* is a member of the variable *callbackRequesterStates*. Fourth, the guard grd4 ensures that the *request* is a member of REQUESTS and the *request* does not exist in the domain *dataAccessRequests*. Fifth, the guard grd5 ensures that adding an ordered pair (request $\mapsto$ dataSubjectConsent) into variable *dataAccessRequests* must satisfy the invariant inv7. Whenever all guards are valid, action act1 adds an ordered pair (request $\mapsto$ dataSubjectConsent) to the *dataAccessRequests*.

The CallbackResponder event (Listing 5) handles a callback from the blockchain to the responder. When the responder receives a callback, it will respond to a request for accessing the personal data within the retention period. The guards

CallbackResponder $\widehat{=}$

Any $oraclizeFee, request$ **Where**

$grd1 : this \in \text{dom}(balanceOf) \land oraclizeFee \in \mathbb{N} \land oraclizeFee \leq balanceOf(this)$

$grd2 : balanceOf \lhd\!\!+ \{this \mapsto balanceOf(this) - oraclizeFee\} \in addresses \rightarrow \mathbb{N}$

$grd3 : request \in \text{dom}(dataAccessRequests) \land request \notin callbackResponderStates$

$grd4 : dataAccessRequests(request) \in dom(dataSubjectConsents) \land$
$\quad dataSubjectConsents(dataAccessRequests(request)) = TRUE$

Then

$act1 : balanceOf := balanceOf \lhd\!\!+ \{this \mapsto balanceOf(this) - oraclizeFee\}$

$act2 : callbackResponderStates := callbackResponderStates \cup \{request\}$

End

Listing 5: The CallbackResponder event.

SubmitResponse $\widehat{=}$

Any $consentExpired, request, response$ **Where**

$grd1 : consentExpired \in BOOL \land consentExpired = FALSE$

$grd2 : dataSubjectConsent \in dom(dataSubjectConsents) \land dataSubjectConsents(dataSubjectConsent) = TRUE$

$grd3 : request \in callbackResponderStates$

$grd4 : response \in RESPONSES \land response \notin dom(dataAccessResponses)$

$grd5 : dataAccessResponses \lhd\!\!+ \{request \mapsto response\} \in RESPONSES \rightarrowtail REQUESTS$

Then

$act1 : dataAccessResponses(response) := request$

End

Listing 6: The SubmitResponse event.

are defined with four preconditions. The first two guards are the same as in the CallbackRequester event. Additionally, we declared the guard grd3 to ensure that the *request* is a member of the domain *dataAccessRequests*, and the request does not exist in the *callbackResponderStates*. Finally, through the guard grd4, we specified that the *dataAccessRequests(request)* as a *dataSubjectConsent* is a member of the domain *dataSubjectConsents*, and the range of the *dataSubjectConsents(dataAccessRequests(request))* as a boolean is TRUE. Whenever all guards are valid, action act1 charges *oraclizeFee* from the *balanceOf(this)*, and action act2 adds the *request* to the *callbackResponderStates*.

The SubmitResponse event (Listing 6) is used to handle the request's response of a requester. Before returning the response to the requester, the event must check the following constraints: 1) the consent has not expired, 2) the consent has not been withdrawn, and 3) the response ID has not been submitted. Based on these constraints, guards are defined with five preconditions. The first two guards are the same as in the SubmitRequest event. Additionally, we declared guards grd3 to ensure that the *request* is a member of the variable *callbackResponderStates* and grd4 to ensure that the response is a member of RESPONSES and the *response* does not exist in the domain *dataAccessResponses*. Finally, the last guard grd5 ensures that adding an ordered pair (response $\mapsto$ request) into variable *dataAccessResponses* must satisfy the invariant inv9. Whenever all guards are valid, action act1 adds an ordered pair (response $\mapsto$ request) to the *dataAccessResponses*.

The CallbackDataTransfer event (Listing 7) is used to handle a callback from the blockchain to trigger the responder for data transfer. When the responder receives a callback, it will start to transfer personal data to the requester directly. The guards are defined with four preconditions. The first two guards are the same as in the CallbackRequester event. Additionally, we declared the guard grd3 to ensure that the *dataAccessResponses*, and the response does not exist in the *callbackDataTransferStates*. By means of the guard grd4, we specified that the *dataAccessResponses(response)* as a *request* is a member of the domain *dataAccessRequests*, *dataAccessRequests(dataAccessResponses(response))* as a *dataSubjectConsent* is member of the domain *dataSubjectConsents*, and the range of the *dataSubjectConsents(dataAccessRequests(dataAccessResponses(response)))* as a boolean is TRUE. Whenever all guards are valid, action act1 charges *oraclizeFee* from the *balanceOf(this)*, and action act2 adds the *response* to the *callbackDataTransferStates*.

CallbackDataTransfer $\widehat{=}$

Any $oraclizeFee, request$ **Where**

$grd1 : this \in \text{dom}(balanceOf) \land oraclizeFee \in \mathbb{N} \land oraclizeFee \leq balanceOf(this)$

$grd2 : balanceOf \lhd\!\!+ \{this \mapsto balanceOf(this) - oraclizeFee\} \in addresses \rightarrow \mathbb{N}$

$grd3 : response \in \text{dom}(dataAccessResponses) \land response \notin callbackDataTransferStates$

$grd4 : dataAccessResponses(response) \in dom(dataAccessRequests) \land$
$dataAccessRequests(dataAccessResponses(response)) \in dom(dataSubjectConsents) \land$
$dataSubjectConsents(dataAccessRequests(dataAccessResponses(response))) = TRUE$

Then

$act1 : balanceOf := balanceOf \lhd\!\!+ \{this \mapsto balanceOf(this) - oraclizeFee\}$

$act2 : callbackDataTransferStates := callbackDataTransferStates \cup \{response\}$

End

Listing 7: The CallbackDataTransfer event.

TransferData $\widehat{=}$

Any $responder, dataSubject, consent, response$ **Where**

$grd1 : response \in callbackDataTransferStates \land response \in \text{dom}(dataAccessResponses) \land$
$response \notin dom(dataTransferStates)$

$grd2 : consent \in \text{dom}(dataFields)$

$grd3 : \exists x \cdot x \in dataAccessRequests[\{dataAccessResponses(response)\}] \land$
$x = responder \mapsto dataSubject \mapsto consent \land$
$responder \mapsto dataSubject \mapsto consent \in \text{dom}(dataSubjectConsents) \land$
$dataSubjectConsents(x) = TRUE$

$grd4 : encryptedData \lhd\!\!+ \{response \mapsto \{dataSubject\} \times dataFields(consent)\} \in$
$RESPONSES \nrightarrow \mathbb{P}(DATA\_SUBJECTS \times FIELDS)$

$grd5 : dataTransferStates \lhd\!\!+ \{response \mapsto TRUE\} \in RESPONSES \nrightarrow BOOL$

Then

$act1 : encryptedData(response) := \{dataSubject\} \times dataFields(consent)$

$act2 : dataTransferStates(response) := TRUE$

End

Listing 8: The TransferData event.

The TransferData event (Listing 8) transmits personal data from the responder to the requester via an off-chain channel. Before the personal data is transmitted, all constraints must be satisfied. The guards are defined with five preconditions. First, the guard grd1 ensures that the *response* is a member of the *callbackDataTransferStates* and the domain *dataAccessResponses*, and the *response* does not exist in the domain *dataTransferStates*. Second, the guard grd2 ensures that *consent* is a member of the domain *dataFields*. Third, the guard grd3 ensures that the data subject's consent is active and exists in the variables *dataAccessRequests*. Fourth, the guard grd4 ensures that adding an ordered pair (response $\mapsto$ {dataSubject} $\times$ dataFields(consent)) into variable *encryptedData* must satisfy the invariant inv11. Fifth, the guard grd5 ensures that adding an ordered pair (response $\mapsto$ TRUE) into variable *dataTransferStates* must satisfy the invariant inv12. Whenever all guards are valid, the action act1 adds an ordered pair (response $\mapsto$ {dataSubject} $\times$ dataFields(consent)) to the *encryptedData*, and action act2 adds an ordered pair (response $\mapsto$ TRUE) to the *dataTransferStates*.

The RevokeConsent event (Listing 9) is fired when a data subject requests to withdraw his/her consent. The guards are defined with two preconditions. First, the guard grd1 ensures that *dataSubjectConsent* is a member of the domain *dataSubjectConsents* and the active status of the *dataSubjectConsents(dataSubjectConsent)* is TRUE. The second guard grd2

RevokeConsent $\widehat{=}$

Any *dataSubjectConsent* **Where**

$grd1 : dataSubjectConsent \in \text{dom}(dataSubjectConsents) \land$

$dataSubjectConsents(dataSubjectConsent) = TRUE$

$grd2 : dataSubjectConsents \lhd\!\!+ \{dataSubjectConsent \mapsto FALSE\} \in$

$(PARTICIPANTS \times DATA\_SUBJECTS \times CONSENTS) \nrightarrow BOOL$

Then

$act1 : dataSubjectConsents(dataSubjectConsent) := FALSE$

End

Listing 9: The RevokeConsent event.

RenewConsent $\widehat{=}$

Any *dataSubjectConsent* **Where**

$grd1 : dataSubjectConsent \in \text{dom}(dataSubjectConsents)$

$dataSubjectConsents(dataSubjectConsent) = FALSE$

$grd2 : dataSubjectConsents \lhd\!\!+ \{dataSubjectConsent \mapsto TRUE\} \in$

$(PARTICIPANTS \times DATA\_SUBJECTS \times CONSENTS) \nrightarrow BOOL$

Then

$act1 : dataSubjectConsents(dataSubjectConsent) := TRUE$

End

Listing 10: The RenewConsent event.

InsufficientBalance $\widehat{=}$

Any *oraclizeFee*, *dataSubjectConsent*, *request*, *response* **Where**

$grd1 : this \in \text{dom}(balanceOf) \land oraclizeFee \in \mathbb{N} \land oraclizeFee > balanceOf(this)$

$grd2 : dataSubjectConsent \in dom(dataSubjectConsents) \land$

$dataSubjectConsents(dataSubjectConsent) = TRUE$

$grd3 : (dataSubjectConsent \notin callbackRequesterStates) \lor$

$(request \mapsto dataSubjectConsent \in dataAccessRequests \land request \notin callbackResponderStates) \lor$

$(response \mapsto request \in dataAccessResponses \land response \notin callbackDataTransferStates)$

Then

skip

End

Listing 11: The InsufficientBalance event.

ensures that when updating the FALSE to the *dataSubjectConsents(dataSubjectConsent)*, the invariant inv3 must be satisfied. Whenever all guards are valid, action act1 adds the FALSE to the *dataSubjectConsents(dataSubjectConsent)*.

The RenewConsent event (Listing 10) is fired when a data subject requests to renew his/her consent. The guards are defined with two preconditions. First, the guard grd1 ensures that *dataSubjectConsent* is a member of the domain *dataSubjectConsents* and the active status of the *dataSubjectConsents(dataSubjectConsent)* is FALSE. The guard grd2 ensures that when updating the TRUE to the *dataSubjectConsents(dataSubjectConsent)*, the invariant inv3 must be satisfied. Whenever all guards are valid, the action act1 adds TRUE to the *dataSubjectConsents(dataSubjectConsent)*.

The InsufficientBalance event (Listing 11) handles the insufficient balance within a smart contract. The insufficient balance occurs when a smart contract's balance is too low to cover fees. The guards are defined with three preconditions. First, the guard grd1 ensures that the constant this is a member of the domain *balanceOf*, the *oraclizeFee* is a member of the set of

CheckConsentExpiration $\widehat{=}$

Any $consentExpired, dataSubjectConsent$ **Where**

$grd1 : consentExpired \in BOOL \wedge consentExpired = TRUE$

$grd2 : dataSubjectConsent \in dom(dataSubjectConsents) \wedge$

$dataSubjectConsents(dataSubjectConsent) = TRUE$

$grd3 : dataSubjectConsents \leftarrow \{dataSubjectConsent \mapsto FALSE\} \in$

$PARTICIPANTS \times DATA\_SUBJECTS \times CONSENTS \nrightarrow BOOL$

Then

$act1 : dataSubjectConsents(dataSubjectConsent) := FALSE$

End

Listing 12: The CheckConsentExpiration event.

UnauthorizedAccess $\widehat{=}$

Any $dataSubjectConsent$ **Where**

$grd1 : dataSubjectConsent \in dom(dataSubjectConsents) \wedge$

$dataSubjectConsents(dataSubjectConsent) = FALSE$

Then

$skip$

End

Listing 13: The UnauthorizedAccess event.

Table 4
The summary of proof statistics by the Rodin platform for the proposed state machine based on the Event-B model.

Machine name	Number of proof obligations	Automatic (%)	Manual (%)
DSSM	42	42(100%)	0(0%)

natural numbers, and the *oraclizeFee* must be greater than *balanceOf(this)* Second, the guard grd2 ensures that *dataSubjectConsent* is a member of the domain *dataSubjectConsents* and the active status of the *dataSubjectConsents(dataSubjectConsent)* is TRUE. Third, the guard grd3 ensures that insufficient balance occurs in callback events. Whenever all of the guards are valid, the process ends.

The CheckConsentExpiration event (Listing 12) is used to handle when data subjects' consent is expired. The guards are defined with three preconditions. First, the guard grd1 ensures that the *consentExpired* is a member of the boolean and *consentExpired* is TRUE. Second, the guard grd2 ensures that *dataSubjectConsent* is a member of the domain *dataSubjectConsents* and the active status of the *dataSubjectConsents(dataSubjectConsent)* is TRUE. Third, the guard grd3 ensures that when updating the FALSE to the *dataSubjectConsents(dataSubjectConsent)*, the invariant inv3 must be satisfied. Whenever all guards are valid, the action act1 adds FALSE to the *dataSubjectConsents(dataSubjectConsent)*.

The UnauthorizedAccess event (Listing 13) is used to handle when there is a request to access the data of a data subject, but the data subject's consent has been revoked or expired. The guard grd1 ensures that *dataSubjectConsent* is a member of the domain *dataSubjectConsents* and the active status of the *dataSubjectConsents(dataSubjectConsent)* is FALSE. Whenever the guard is valid, the process ends.

6. Model evaluation in Event-B

The DSSM was formalized with Event-B, and its correctness was verified using the Rodin Platform. The Rodin Platform produces and discharges a set of POs automatically or manually to ensure that all events preserve all invariants. The resulting model (Table 4) demonstrates that the DSSM was proved automatically by Atelier B provers. As a result, there are no invariant violations or deadlocks found. The proposed model is available for the public at https://github.com/cucpbioinfo/BlockchainBasedDataSharing.

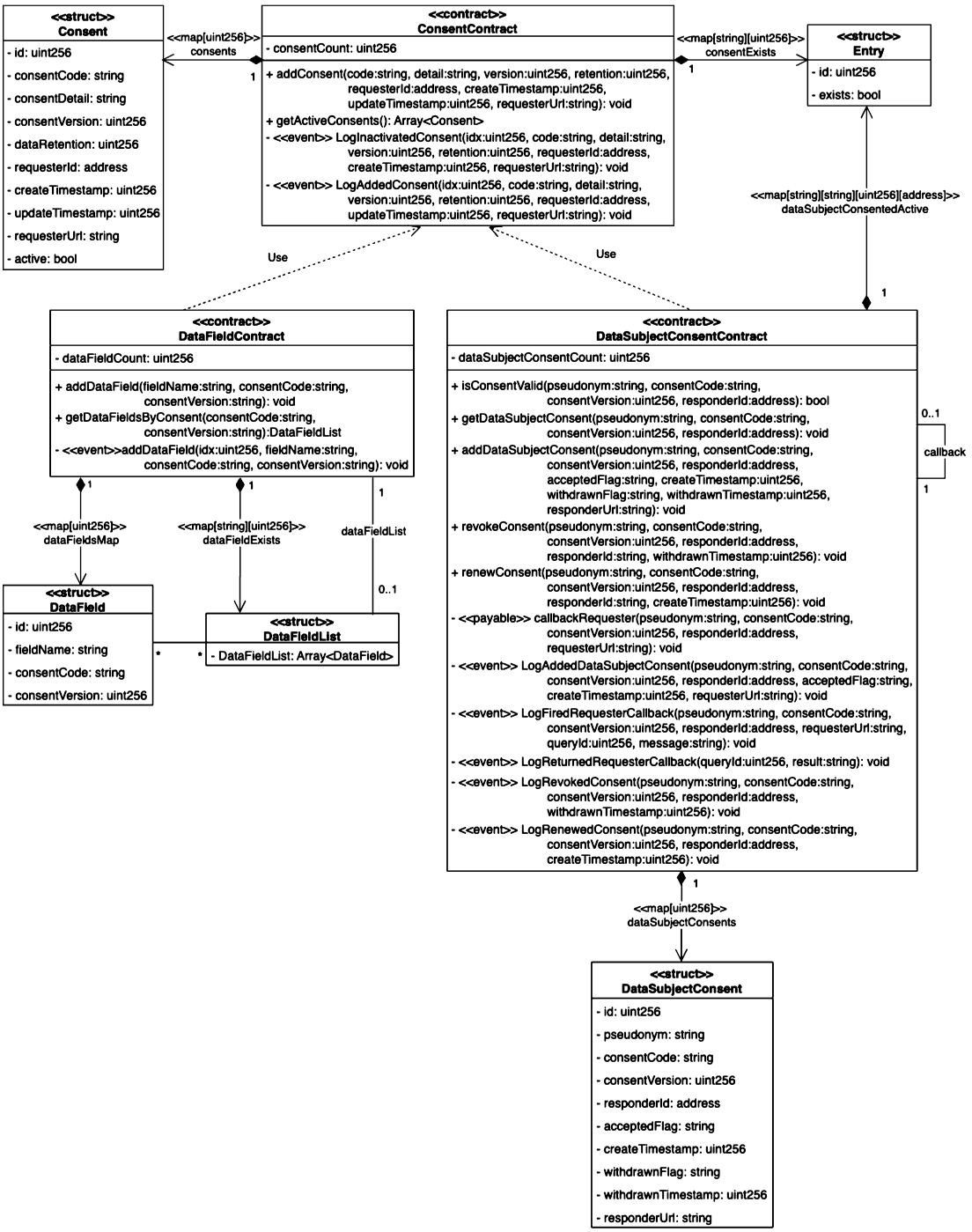


Fig. 4. Class diagram resulted from mapping the proposed model in Event-B to code for supporting consent management in the context of data sharing.

7. Event-B model transformation to class diagram

The proposed model constructed by Event-B assists developers as a guideline in applying consent management functionality among distributed services based on blockchain technology. In an object-oriented approach, a class diagram depicts a static view of a system, which is described by modeling its classes, attributes, operations, and associations. Moreover, a class diagram makes it easier for developers to understand how to implement smart contracts to support consent management. Here is an example of how to transform our Event-B model into a class diagram. First, identify a system's classes that appear in static variables of Event-B (e.g., carrier sets, constants, and variables). Second, identify a system's class associations among

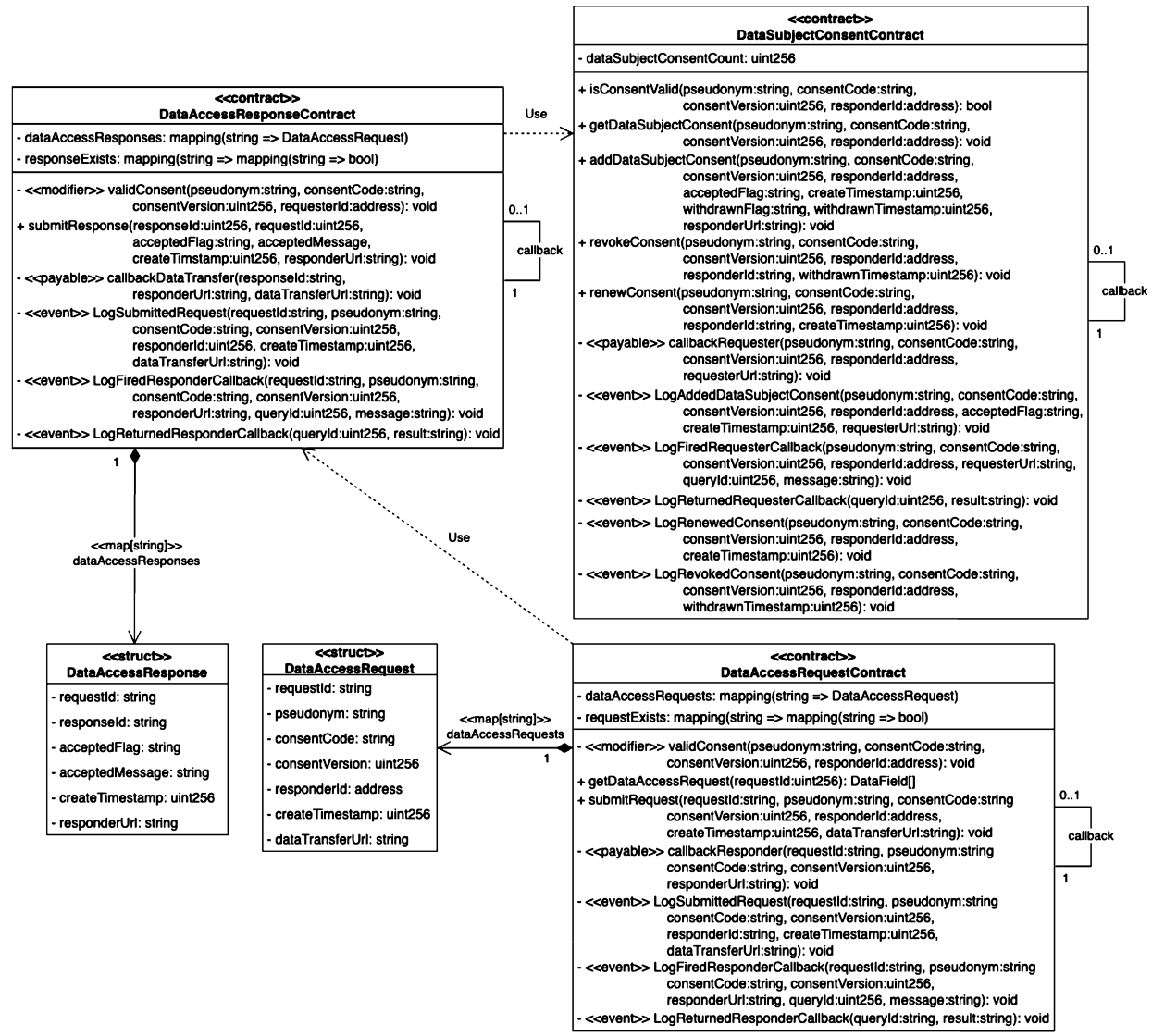


Fig. 5. Class diagram continued from the previous diagram (Fig. 4), which demonstrates how to transform the proposed model in Event-B for supporting request-response interactions.

itself or other sets that appear in invariants. Third, identify a system's operations in classes. Besides, each of the transitions has guard conditions, and it can be fired when the guard conditions are evaluated to be true, then an event occurs. Each guard condition represents a precondition based on state variables inside a method within classes. Figs. 4 and 5 show the class diagram designed based on Ethereum smart contracts using Solidity.

The static variables in Event-B can be mapped to concrete classes, which are divided into two groups: 1) the classes used in consent management functionality, e.g., *Consent*, *ConsentContract*, *DataSubjectConsent*, *DataSubjectConsentContract*, as shown in Fig. 4, and 2) the classes used in request-response interactions between services, e.g., *DataAccessRequest*, *DataAccessRequestContract*, *DataAccessResponse*, *DataAccessResponseContract*, as shown in Fig. 5. The set of DATA_SUBJECTS and CONSENTS represents data subjects and consents under GDPR, respectively. According to GDPR, the system requires gaining data subjects' consent before processing data. Therefore, we first defined structs (i.e., user-defined data types that obtain related data items, probably of different data types); for example, Consent is used to hold a set of properties (e.g., consentCode, consentVersion, consentDetail, dataRetention (in days), createTimestamp, requesterUrl). Within our proposed, the system should inform data subjects which piece of personal data is being used. So, we created *DataField* to hold pre-defined data fields upon the requesters' consent and used it to specify the personal data to be transferred. Moreover, *DataSubjectConsent* must be created to hold the relationship between data subjects and requesters' consent. Second, we defined contracts (i.e., classes that obtain state variables and methods); for example, *ConsentContract* is to provide the addConsent() method, which is mapped to the AddConsent event. In the AddDataSubjectConsent event, we further cre-

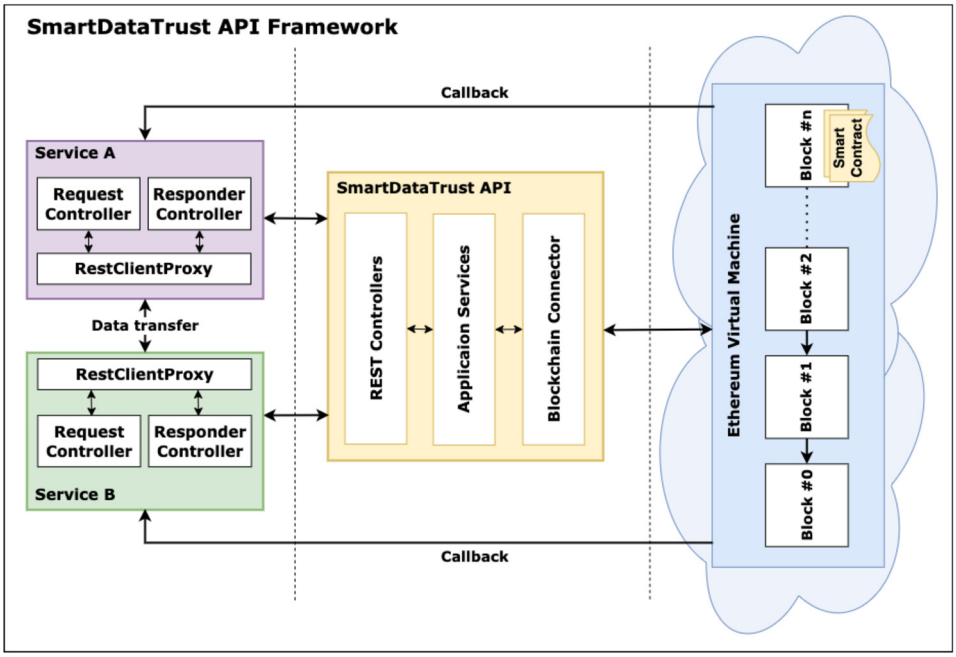


Fig. 6. Overview of SmartDataTrust API framework.

ated *DataSubjectConsentContract* to obtain the addDataSubjectConsent() method. In addition, we added isConsentValid() as a common method to check whether consent is expired or withdrawn. As for the request-response interactions, we created *DataAcessRequestContract* and *DataAcessResponseContract* to obtain *DataAcessRequest* and *DataAcessResponse* structs, respectively. For the SubmitRequest and SubmitResponse events, the model is required to check whether consent is valid. Hence, we then added the isConsentValid() method into these two contracts by calling *DataSubjectConsentContract*.isConsentValid(). According to the class diagram in Figs. 4 and 5, our proposed model was designed for common use and can be applied to any business model. To allow the developers to quickly adopt the model, we developed SmartDataTrust that implemented smart contracts based on these class diagrams and exposed a REST API to interact with the blockchain. The requester and responder services only need to focus on implementing a REST API for consuming SmartDataTrust API and providing the callback URLs made by the blockchain.

8. SmartDataTrust implementation

The SmartDataTrust API is a middleware that interacts with smart contracts live on the Ethereum blockchain by exposing REST services to the outside world (Fig. 6). Implementing this API aims to provide a set of consent functionality for requester and responder services, which minimizes the effort of incorporating GDPR-compliant consent management in their interacting services. Moreover, it supports scalability by separating configuration from code in the YAML format (i.e., config.yaml), which is easily configured to deploy as Docker containers [50] with Kubernetes [51]. The API was designed based on a three-layer architecture [52] partitioned into REST controllers (i.e., consent_controller.py, data_subject_controller.py, data_access_request_controller.py, data_access_response_controller.py), application services (i.e., consent_service.py, data_field_service.py, data_subject_service.py, data_access_request_service.py, data_access_response_service.py), and the blockchain connector (i.e., blockchain_connector.py). The REST controllers handle incoming HTTP requests from requester and responder services and pass them through the application services. As for application services, they encapsulate data validation and conversion. Finally, the blockchain connector uses web3 frameworks [53] (e.g., Web3.py, Ethers.js, Infura API) for connecting smart contracts on the Ethereum blockchain through their contacts' addresses and contracts' schema files, which are configured in config.yaml.

In smart contract development, we first plug Truffle Suite [54] into SmartDataTrust API for building and deploying smart contracts on the Ethereum blockchain. Second, we implement smart contracts with Solidity followed by the class diagram, as shown in Figs. 4 and 5. Third, we deploy smart contracts using Truffle's command (i.e., truffle migrate). After successful deployment, Truffle Suits generates the contracts' address and contracts' schema in JSON format files. Fourth, we configure the contracts' address and schema path into config.yaml. Finally, we start the Python REST API.

Unfortunately, a smart contract is an immutable program. Once it is deployed on the blockchain, it preserves a new address. However, the multiple times of deployments of the smart contract lead to difficulty managing addresses and increasing execution time. We then designed reusable smart contracts to keep only states of data subjects' consent and request-response interactions between services. To create a callback URL outside the blockchain, we use blockchain oracles

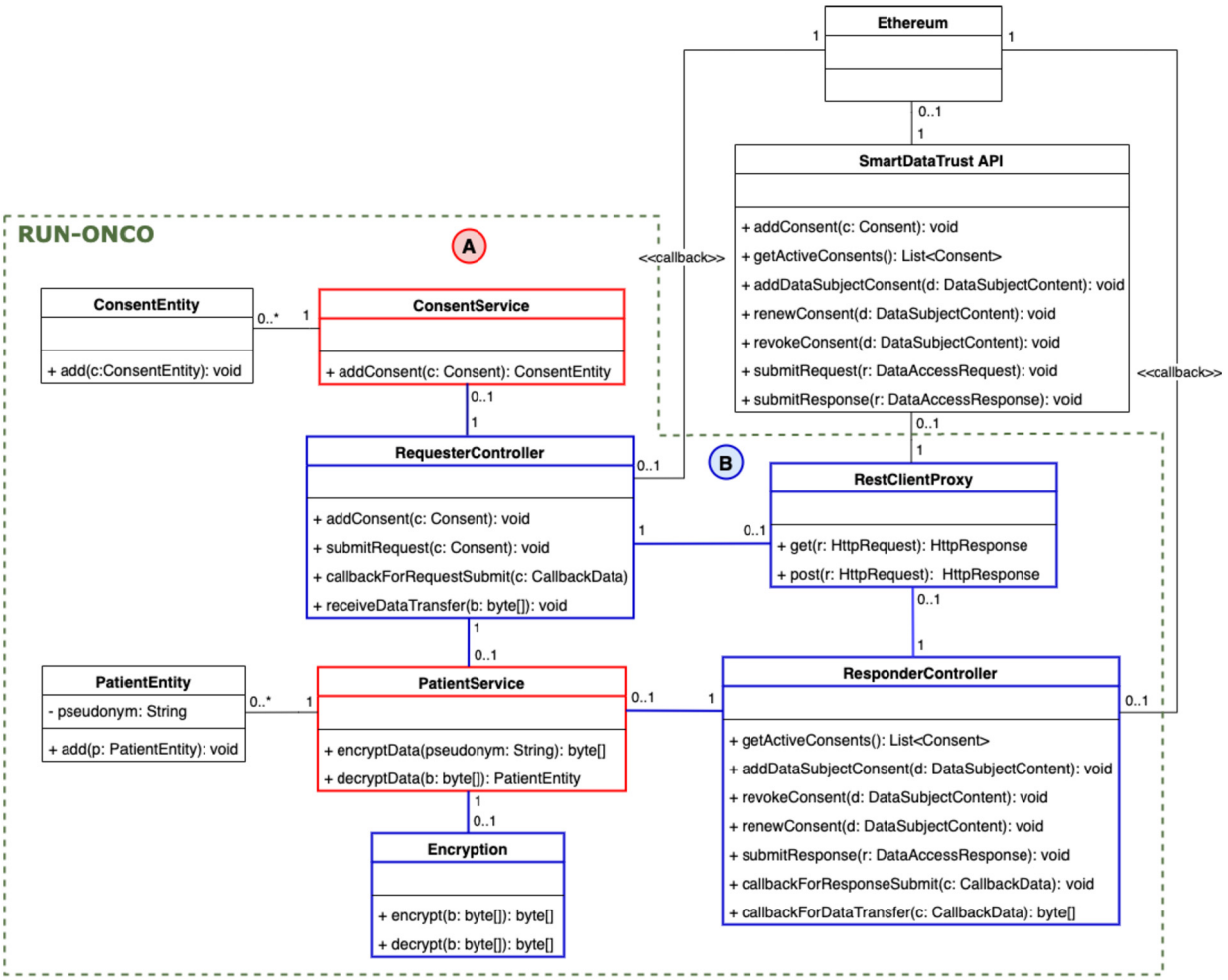


Fig. 7. Class diagram demonstrating how a software platform for cancer precision medicine handles GDPR-compliant blockchain-based consent management in data sharing. (A) relevant classes needed to be enhanced to support data sharing. (B) new classes added to RUN-ONCO for supporting managed consent into the blockchain and handling the requester and responder callbacks made by the blockchain.

[55], e.g., Provable, Chainlink, and Astraea. In particular, we choose Provable for integrating into smart contracts because it is easy to implement and support dynamic data retrieval from trusted sources in large-scale applications.

To enhance an existing system integrated with the SmartDataTrust API, we demonstrate via a software platform for cancer precision medicine called RUN-ONCO [56]. RUN-ONCO allows users (i.e., oncologists, nurses, and researchers) to manage and create their own data analyzes to examine clinical, biospecimen, and genetic data, which assists oncologists in making specific treatment plans for individual cancer patients based on their genetics. To engage in research on cancer precision medicine, we need more patient data to help discover how to improve patient outcomes, such as genetic data and drug response. Therefore, we need to enhance RUN-ONCO to enable data sharing to exchange health data across organizations and between services. We then divided services into two types: 1) the service which manages its own patients' data, e.g., health information systems, and 2) the service which does not contain any patients' data, e.g., third-party API. RUN-ONCO and other services only focused on implementing a REST API for consuming the SmartDataTrust API to manage consent requests/responses on the blockchain and handling the requester and responder callbacks made by the blockchain. To enhance RUN-ONCO support consent management in data sharing (Figs. 7A and 7B), by following DSSM, we first need to alter the *ConsentService* class by adding the addConsent() method. Second, we need to add the encryptData() and decryptData() methods into the *PatientService* class to support secure data transfer between services. As for any service, it can be either a requester or a responder. We then create *RequesterController* and *ResponderController* classes following the available services in the SmartDataTrust API, and to handle API calls and HTTPS GET/POST requests among blockchain; we create *RestClientProxy* class.

9. Conclusion

Data sharing enables organizations to exchange data across their ecosystem to develop new and improved products or services. On the other hand, data sharing brings a great deal of academic research. For example, medical research studies involve individuals' health information to help patients live longer and healthier, creating high value for society. However, sharing individuals' data held by any sector should comply with adequate provisions to safeguard entrusted information. There are numerous publications on data sharing using blockchain technology, but instead, they are focused on a human-centric approach that needs data subjects or third-party organizations to first store personal data into off-chain storage servers and then can begin to request access to personal data. The advantage of our proposed model is providing a solution that enables the two or more different platforms to share data without using off-chain storage servers. Therefore, we proposed a privacy-enabled formal data-sharing model, DSSM, incorporating PbD to ensure compliance with GDPR requirements. We used the Event-B method to prove the correctness of the proposed model, which combines the formalization of mathematical proofs based on first-order logic and set theory. In addition, the proposed model was mapped into smart contracts' code and implemented as a prototype SmartDataTrust platform providing essential technical insights into implementing consent management functionality among distributed services.

In future work, we will focus on incentive mechanisms for our proposed model to evaluate data subjects' compensation cost against data sharing cost and balance incentives for all actors. Moreover, a smart contract on the Ethereum blockchain by default has no owner, meaning that once the smart contract is deployed, anyone can access it through the network. So, we will explore access control design patterns for authorization checking in smart contract execution based on the account addresses whitelist.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] European Commission, Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance), https://eur-lex.europa.eu/eli/reg/2016/679/oj, 2016.
[2] P.V. Kakarlapudi, Q.H. Mahmoud, A systematic review of blockchain for consent management, Healthcare 9 (2) (2021), https://doi.org/10.3390/healthcare9020137, https://www.mdpi.com/2227-9032/9/2/137.
[3] V. Jaiman, V. Urovi, A consent model for blockchain-based health data sharing platforms, IEEE Access 8 (2020) 143734–143745, https://doi.org/10.1109/ACCESS.2020.3014565.
[4] Vargas, J. Camilo, Blockchain-based consent manager for gdpr compliance, in: H. Roßnagel, S. Wagner, D. Hühnlein (Eds.), Open Identity Summit 2019, Gesellschaft für Informatik, Bonn, 2019, pp. 165–170.
[5] H.H. Jung, F.M. Pfister, Blockchain-enabled clinical study consent management, Technol. Innov. Manag. Rev. 10 (2020) 14–24, https://doi.org/10.22215/timreview/1325.
[6] A. Alhazmi, N.A.G. Arachchilage, I'm all ears! Listening to software developers on putting gdpr principles into software development practice, Pers. Ubiquitous Comput. 25 (5) (2021) 879–892, https://doi.org/10.1007/s00779-021-01544-1.
[7] A. Senarath, N.A.G. Arachchilage, Why developers cannot embed privacy into software systems? An empirical investigation, in: Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018, EASE'18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 211–216.
[8] A. Cavoukian, Privacy by Design: The 7 Foundational Principles, revised: January 2011 (August 2009).
[9] A. Cavoukian, Understanding how to implement privacy by design, one step at a time, IEEE Consum. Electron. Mag. 9 (2) (2020) 78–82, https://doi.org/10.1109/MCE.2019.2953739.
[10] L. Alkhariji, N. Alhirabi, M.N. Alraja, M. Barhamgi, O. Rana, C. Perera, Synthesising privacy by design knowledge toward explainable internet of things application designing in healthcare, ACM Trans. Multimed. Comput. Commun. Appl. 17 (2s) (jun 2021), https://doi.org/10.1145/3434186.
[11] B.-J. Koops, R. Leenes, Privacy regulation cannot be hardcoded. A critical comment on the 'privacy by design' provision in data-protection law, Int. Rev. Law Comput. Technol. 28 (2) (2014) 159–171, https://doi.org/10.1080/13600869.2013.801589, arXiv:https://doi.org/10.1080/13600869.2013.801589.
[12] M. Finck, Blockchains and data protection in the European Union, Soc. Sci. Res. Netw. (2017), https://doi.org/10.2139/ssrn.3080322.
[13] M.K.S. Suripeddi, P. Purandare, Blockchain and gdpr - a study on compatibility issues of the distributed ledger technology with gdpr data processing, J. Phys. Conf. Ser. 1964 (4) (2021) 042005, https://doi.org/10.1088/1742-6596/1964/4/042005.
[14] P. Chinnasamy, B. Vinodhini, V. Praveena, C. Vinothini, B.B. Sujitha, Blockchain based access control and data sharing systems for smart devices, J. Phys. Conf. Ser. 1767 (1) (2021) 012056, https://doi.org/10.1088/1742-6596/1767/1/012056.
[15] X. Wang, Q. Li, Design and implementation of a data sharing model for improving blockchain technology, Adv. Multimed. 2022 (Jan 2022), https://doi.org/10.1155/2022/4578525.
[16] T.K. Agrawal, J. Angelis, W.A. Khilji, R. Kalaiarasan, M. Wiktorsson, Demonstration of a blockchain-based framework using smart contracts for supply chain collaboration, Int. J. Prod. Res. 61 (5) (2023) 1497–1516, https://doi.org/10.1080/00207543.2022.2039413.
[17] J.-R. Abrial, Modeling in Event-B: System and Software Engineering, Cambridge University Press, 2010.
[18] S. Chong, J. Guttman, A. Datta, A. Myers, B. Pierce, P. Schaumont, T. Sherwood, N. Zeldovich, Report on the NSF Workshop on Formal Methods for Security, 2016.

[19] M.C. Tschantz, J.M. Wing, Formal methods for privacy, in: A. Cavalcanti, D.R. Dams (Eds.), FM 2009: Formal Methods, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 1–15.
[20] J.-R. Abrial, M. Butler, S. Hallerstede, T.S. Hoang, F. Mehta, L. Voisin Rodin, An open toolset for modelling and reasoning in Event-B, Int. J. Softw. Tools Technol. Transf. 12 (6) (2010) 447–466, https://doi.org/10.1007/s10009-010-0145-y.
[21] C. Daudén-Esmel, J. Castellà-Roca, A. Viejo, J. Domingo-Ferrer, Lightweight blockchain-based platform for gdpr-compliant personal data management, in: 2021 IEEE 5th International Conference on Cryptography, Security and Privacy (CSP), 2021, pp. 68–73.
[22] M.M. Merlec, Y.K. Lee, S.-P. Hong, H.P. In, A smart contract-based dynamic consent management system for personal data usage under gdpr, Sensors 21 (23) (2021), https://doi.org/10.3390/s21237994, https://www.mdpi.com/1424-8220/21/23/7994.
[23] K. Rantos, G. Drosatos, K. Demertzis, C. Ilioudis, A. Papanikolaou, A. Kritsas, Advocate: a consent management platform for personal data processing in the iot using blockchain technology, in: J.-L. Lanet, C. Toma (Eds.), Innovative Security Solutions for Information Technology and Communications, Springer International Publishing, Cham, 2019, pp. 300–313.
[24] A. Azaria, A. Ekblaw, T. Vieira, A. Lippman, Medrec: using blockchain for medical data access and permission management, in: 2016 2nd International Conference on Open and Big Data (OBD), 2016, pp. 25–30.
[25] C. Hu, C. Li, G. Zhang, Z. Lei, M. Shah, Y. Zhang, C. Xing, J. Jiang, R. Bao, Crowdmed-ii: a blockchain-based framework for efficient consent management in health data sharing, World Wide Web 25 (3) (2022) 1489–1515, https://doi.org/10.1007/s11280-021-00923-1.
[26] M. Shah, C. Li, M. Sheng, Y. Zhang, C. Xing, Crowdmed: a blockchain-based approach to consent management for health data sharing, in: H. Chen, D. Zeng, X. Yan, C. Xing (Eds.), Smart Health, Springer International Publishing, Cham, 2019, pp. 345–356.
[27] S. Rouhani, L. Butterworth, A.D. Simmons, D.G. Humphery, R. Deters, Medichaintm: a secure decentralized medical data asset management system, in: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2018, pp. 1533–1538.
[28] Medicalchain whitepaper 2.1, http://medicalchain.com/en/whitepaper, 2018. (Accessed 22 March 2022).
[29] R.R. Agarwal, D. Kumar, L. Golab, S. Keshav, Consentio: managing consent to data access using permissioned blockchains, in: 2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2020, pp. 1–9.
[30] C.C. Agbo, Q.H. Mahmoud, Design and implementation of a blockchain-based e-health consent management framework, in: 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2020, pp. 812–817.
[31] N. Mamo, G.M. Martin, M. Desira, B. Ellul, J.-P. Ebejer, Dwarna: a blockchain solution for dynamic consent in biobanking, Eur. J. Hum. Genet. 28 (5) (2020) 609–626, https://doi.org/10.1038/s41431-019-0560-9.
[32] D. Ameyed, F. Jaafar, F. Charette-Migneault, M. Cheriet, Blockchain based model for consent management and data transparency assurance, in: 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2021, pp. 1050–1059.
[33] S. Gürses, C. Troncoso, C. Diaz, Engineering privacy by design, in: Proceedings of the 4th International Conference on Computers, vol. 14, 2011, p. 25.
[34] S.-S. Jung, S.-J. Lee, I.-C. Euom, Delegation-based personal data processing request notarization framework for gdpr based on private blockchain, Appl. Sci. 11 (22) (2021), https://doi.org/10.3390/app112210574, https://www.mdpi.com/2076-3417/11/22/10574.
[35] M.B. Blake, I. Saleh, Formal methods for preserving privacy for big data extraction software, in: NSF Workshop on Big Data Security and Privacy, Dallas, Texas, Sept 2014.
[36] R. Kitchin, Big data and human geography: opportunities, challenges and risks, Dialogues Hum. Geogr. 3 (3) (2013) 262–267, https://doi.org/10.1177/2043820613513388, arXiv:https://doi.org/10.1177/2043820613513388.
[37] D. Miltiadou, S. Pitsios, S. Kasdaglis, D. Spyropoulos, G. Misiakoulis, F. Kossiaras, I. Skarbovsky, F. Fournier, N. Kapsoulis, J. Soldatos, K. Perakis, Leveraging Management of Customers' Consent Exploiting the Benefits of Blockchain Technology Towards Secure Data Sharing, Springer International Publishing, Cham, 2022, pp. 127–155.
[38] Z. Abedjan, N. Boujemaa, S. Campbell, P. Casla, S. Chatterjea, S. Consoli, C. Costa-Soria, P. Czech, M. Despenic, C. Garattini, D. Hamelinck, A. Heinrich, W. Kraaij, J. Kustra, A. Lojo, M.M. Sanchez, M.A. Mayer, M. Melideo, E. Menasalvas, F.M. Aarestrup, E.N. Artigot, M. Petković, D.R. Recupero, A.R. Gonzalez, G.R. Kerremans, R. Roller, M. Romao, S. Ruping, F. Sasaki, W. Spek, N. Stojanovic, J. Thoms, A. Vasiljevs, W. Verachtert, R. Wuyts, Data Science in Healthcare: Benefits, Challenges and Opportunities, Springer International Publishing, Cham, 2019, pp. 3–38.
[39] S. Stalla-Bourdillon, G. Thuermer, J. Walker, L. Carmichael, E. Simperl, Data protection by design: building the foundations of trustworthy data sharing, Data Policy 2 (2020) e4, https://doi.org/10.1017/dap.2020.1.
[40] A.A. Monrat, O. Schelén, K. Andersson, A survey of blockchain from the perspectives of applications, challenges, and opportunities, IEEE Access 7 (2019) 117134–117151, https://doi.org/10.1109/ACCESS.2019.2936094.
[41] N. Ramkumar, G. Sudhasadasivam, K. Saranya, A survey on different consensus mechanisms for the blockchain technology, in: 2020 International Conference on Communication and Signal Processing (ICCSP), 2020, pp. 0458–0464.
[42] S.N. Khan, F. Loukil, C. Ghedira-Guegan, E. Benkhelifa, A. Bani-Hani, Blockchain smart contracts: applications, challenges, and future trends, Peer-to-Peer Netw. Appl. 14 (5) (2021) 2901–2925, https://doi.org/10.1007/s12083-021-01127-0.
[43] D.K. Sharma, S. Pant, M. Sharma, S. Brahmachari, Chapter 13 - cryptocurrency mechanisms for blockchains: models, characteristics, challenges, and applications, in: S. Krishnan, V.E. Balas, E.G. Julie, Y.H. Robinson, S. Balaji, R. Kumar (Eds.), Handbook of Research on Blockchain Technology, Academic Press, 2020, pp. 323–348, https://www.sciencedirect.com/science/article/pii/B9780128198162000137.
[44] A. Lahbib, A. Ait Wakrime, A. Laouiti, K. Toumi, S. Martin, An Event-B based approach for formal modelling and verification of smart contracts, in: L. Barolli, F. Amato, F. Moscato, T. Enokido, M. Takizawa (Eds.), Advanced Information Networking and Applications, Springer International Publishing, Cham, 2020, pp. 1303–1318.
[45] J. Zhu, K. Hu, M. Filali, J.-P. Bodeveix, J.-P. Talpin, H. Cao, Formal simulation and verification of solidity contracts in Event-B, in: 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), 2021, pp. 1309–1314.
[46] J.-R. Abrial, S. Hallerstede, Refinement, decomposition, and instantiation of discrete models: application to Event-B, Fundam. Inform. 77 (1–2) (2007) 1–28.
[47] K. Robinson, A concise summary of the Event-B mathematical toolkit, https://wiki.event-b.org/images/EventB-Summary-refcard.pdf, 2014. (Accessed 11 June 2022).
[48] J.-H. Hoepman, Privacy design strategies, in: N. Cuppens-Boulahia, F. Cuppens, S. Jajodia, A. Abou El Kalam, T. Sans (Eds.), ICT Systems Security and Privacy Protection, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, pp. 446–459.
[49] J. van Rest, D. Boonstra, M. Everts, M. van Rijn, R. van Paassen, Designing privacy-by-design, in: B. Preneel, D. Ikonomou (Eds.), Privacy Technologies and Policy, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, pp. 55–72.
[50] D. Merkel, Docker: lightweight Linux containers for consistent development and deployment, Linux J. 2014 (03) (2014).
[51] S. Dikaleh, O. Sheikh, C. Felix, Introduction to kubernetes, in: Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering, CASCON '17, IBM Corp., USA, 2017, p. 310.
[52] A.O. Ramirez, Three-tier architecture, Linux J. 2000 (75es) (2000) 7–es.
[53] S.K. Panda, S.C. Satapathy, An investigation into smart contract deployment on Ethereum platform using web3.js and solidity using blockchain, in: V. Bhateja, S.C. Satapathy, C.M. Travieso-González, V.N.M. Aradhya (Eds.), Data Engineering and Intelligent Computing, Springer Singapore, Singapore, 2021, pp. 549–561.

[54] Truffle, https://www.trufflesuite.com/truffle, 2020. (Accessed 24 April 2023).
[55] S.K. Ezzat, Y.N.M. Saleh, A.A. Abdel-Hamid, Blockchain oracles: state-of-the-art and research directions, IEEE Access 10 (2022) 67551–67572, https://doi.org/10.1109/ACCESS.2022.3184726.
[56] N. Peyrone, D. Wichadakul, RUN-ONCO: a highly extensible software platform for cancer precision medicine, in: Proceedings of the 2019 6th International Conference on Biomedical and Bioinformatics Engineering, ICBBE '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 142–147.