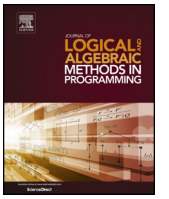
Journal of Logical and Algebraic Methods in Programming 128 (2022) 100789

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp



Formal models for consent-based privacy

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 1 August 2021
Received in revised form 13 June 2022
Accepted 14 June 2022
Available online 20 June 2022

Keywords:
GDPR
Data protection
Privacy by design
Consent management
Formal method
Event-B

ABSTRACT

The General Data Protection Regulation (GDPR) has changed the way businesses handle personal data. The GDPR is a set of conditions within the European Union (EU) law on data protection and privacy. The law requires software systems that store and manage personal data to use only the necessary information ('data minimization') and manage the information fairly and appropriately ('lawfulness, fairness and transparency'). Furthermore, personal data that can lead to direct or indirect identification must be kept safe. Therefore, the risk management of personal data within software mainly depends on the developers' experience. The consent under the GDPR is an agreement between organizations ('data controllers') and individuals ('data subjects'), which provides provisions for protecting personal data. The data controller must gain explicit consent from the data subject before collecting and processing the data. Hence, consent management is an essential component of a software system. This research proposes a set of formal models for consent management that take Privacy by Design (PbD) into account. We used the Event-B method to formalize the proposed models close to a real system. The Rodin platform proved each Event-B model to be corrected and deadlock-free. We also described how developers could transform Event-B models into the actual codes and demonstrated this result by mapping Event-B models into class diagrams. The proposed models meet consent compliance and privacy awareness requirements. In particular, the models cover certain aspects of privacy, including managing the consent of data subjects and controlling authorized access based on the data subject's consent.

1. Introduction

Since the GDPR went into effect on May 25, 2018, it makes people aware of their data protection rights and causes businesses to reevaluate if their software systems are sufficient to protect customers' data. As a developer, it is essential to understand the legal basis of these regulations. However, the developers find the GDPR difficult to interpret and apply to software engineering practices. The lack of clear guidance on how to implement these regulations on software systems leads to risks of confidentiality and privacy breaches. The consequence of privacy cracks will affect business risks of facing heavy fines and penalties mandated by GDPR. The GDPR has significantly altered businesses' handling of personal data to ensure the EU citizens' data rights are protected.

The GDPR is the EU law [1] that protects the right to personal data. It emphasizes the importance of controlling personal data processing and defining roles, e.g., data controller, data processor, that involve. The data controller is responsible for

* 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.2022.100789
2352-2208/

deciding how to collect and process the data (GDPR Articles 4(7) and 24). The data processor is responsible for storing and processing the data (GDPR Articles 4(8) and 28). Regarding consent under GDPR Articles 6(1) and 7, which are divided into four key elements: 1) the consent of the data subject must be given voluntarily, 2) the purpose of processing data must be specific and clear, 3) the data controller must notify the data subject for the purpose, and 4) the data subject gives his/her unambiguous consent. In addition, consent is one of six legal bases to process data according to the GDPR Article 6. Without a data subject's consent, the data controller processes personal data unlawfully or without authorization. Consent management is a software component that handles the entire personal data lifecycle from getting a data subject's consent to the processing of his/her personal data. With consent management capability, software systems are built to manage data subject consent that meets the GDPR requirements or other GDPR-compliance regulations. To ensure that data protection would be embedded in all software systems, the developers should incorporate data privacy by design [2–4]. Privacy by Design (PbD) is an approach that considers data protection as the core functionality of a software system. The advantage of using PbD makes compliance with the GDPR easier to achieve.

The GDPR principles include only the guidelines but not how to design a privacy pattern. PbD is a revolutionary concept that data protection should be promoted as default settings for software systems. In addition, PbD is based on the 7 Foundational Principles representing guidance for developers, which states that 1) it is imperative to design a system that protects personal data, 2) the system must embed privacy as a default, 3) a personal data protection mechanism must be part of the system architecture, 4) the system must keep personal data accurate and secure by the law without compromising the ability of the system, 5) the system must have an appropriate retention period for personal data collection and destruction, 6) the system must announce guidelines for achieving the objective set to those involved such as service providers and recipients to demonstrate transparency and accountability, and 7) the service providers must be respectful and considerate of personal data with the highest level of security. In this paper, we thus propose formal models for consent management by adopting PbD to fulfill compliance obligations from GDPR. In addition, we used the Event-B formal method to guarantee that the system design obligations support the PbD. Formal methods can play an important role in data protection [5,6]. They use mathematical proofs for verifying a software system specification to ensure its correct behaviors. The benefit of using Event-B is that it has an automated tool, called Rodin Platform, that supports the model development and checking. The Rodin Platform is an open-source tool based on the Eclipse IDE, which integrates various plugins, such as model checkers, provers, a proof-obligation generator, etc.

The proposed models covered the consent management operations, including managing data subject consents (Article 4(11) GDPR), determining authorized access based on the data subject's consent (Article 5 GDPR), allowing data subjects to withdraw consents (Articles 17 & 19 GDPR), allowing data subjects to request for their data (Article 20 GDPR), and allowing data subjects to renew consents for keeping their data (Article 5(e) GDPR).

2. Related work

In this section, we discuss similar studies related to managing consent and using formal methods for data protection.

Stouppa & Studer [7] designed a model based on privacy issues in relational databases and ontology-based information systems. The model was defined as an ontology-based query answering for data privacy in terms of entailment. It helps prevent personal data exposure to the public.

Blake & Saleh [8] studied the significance of formal methods to prevent threats of data privacy in the data integration process for big data. Data integration is the process of gathering data from different sources into a data warehouse using a series of Extract, Transform, and Load (i.e., ETL) processes. The authors suggested adding formal methods to validate the conformance of privacy in four main specific areas: 1) Pre-Hadoop process validation, which validates the data preprocessing step and specifies sensitive data, 2) Map-Reduce process validation, which helps reduce the risk of retrieving a massive amount of data and ensures the minimum of personal data being shared between processes, 3) ETL process validation, which checks the privacy compliance policy, then de-identifies the sensitive data and excludes some of them before adding into the data warehouse, and 4) Report testing process, which determines the visibility of sensitive data in the report forms according to specific purposes.

Matwin et al. [9] proposed an approach in privacy-preserving data mining using formal methods for proving the properties of data-mining programs. The authors transformed the logic of data-mining programs into logic expressions of theorem provers. This makes it easier to identify privacy properties. The model was declared with a set of permissions to restrict program properties from accessing personal data according to the data subject's permissions. This solution helps prevent privacy violations from executing the program and uses Coq [10] to prove the correctness.

Consent is an agreement between a data controller and a data subject, which provides a legal basis to process personal data [11]. According to the GDPR, it is unlawful to process personal data without the data subject's consent. There are several papers that included consent as a part of the system design. Tokas & Owe [12] worked on a formal framework for consent management of individuals. The framework allows data subjects to change their privacy settings through a distributed system. The authors declared a set of ordered pairs (subject, purpose), called data tagging, which describes the relationship between a data subject and a specific purpose according to the policy specification. The policy specification represents a set of rules for limiting access to sensitive data. The framework also defined interfaces and classes to hold data subjects' privacy settings. To integrate consent management into the system, the developers need to implement these

Table 1
List of proposed state machines and GDPR articles they covered.

Machine name	GDPR article	Summary
RPSM	Article 4(1)	Personal data is any information (e.g., full name, social security number, medical records) that can directly or indirectly identify a person.
	Article 4(11)	The data subject's consent indicates that the data subject gives his/her unambiguous consent.
	Article 5	Any processing of personal data must be done following the GDPR's six legal bases in Article 5, including 1) personal data processing must always be legitimate, explicit, and genuine with data subjects, 2) personal data must be collected and processed for a specific purpose, 3) personal data must be collected only the data necessary upon the consent, 4) personal data must be accurate and kept up-to-date, 5) personal data must be collected for a specific purpose upon expiration of the time period, and 6) personal data must be accurate and consistent over its entire lifecycle.
WASM	Article 7(3)	Data subjects have the right to revoke their consent for processing at any time.
	Article 17 and 19	Data subjects have the right to request to erase their personal data.
PASM	Article 20	Data subjects have the right to request a portable copy of their personal data.
CRSM	Article 6(1)	Data subjects need to sign consent before processing their personal data.

interfaces and classes. The framework guaranteed that each request for accessing personal data complies with the current consent policies.

Basik & Freytag [13] focused on privacy in healthcare and used Business Process Model and Notation (BPMN) for modeling clinical workflows. The authors defined privacy preferences based on GDPR principles by creating a formal model, which transformed the workflow-based Newborn Screening procedure without privacy detection of children's rights into a workflow-based with privacy detection. Abe & Simpson [14] designed a formal model to prevent unauthorized access to shared data between processes in distributed systems. To protect personal data that are shared between processes, the authors formalized the model with three major constraints: 1) a mechanism of role-based access control (RBAC) [15] policies to protect resources, 2) a source of personal data to be processed, and 3) the relationship between a process identifier and its permissions during data processing.

Ni et al. [16] proposed Privacy-aware Role Based Access Control (P-RBAC), which extended the RBAC model. The RBAC is a mechanism that restricts all users on the system to perform only specified actions on resources but does not support enforcing data protection regulations. The P-RBAC model was defined as a many-to-many relationship between role assignments and permission assignments (PAs). This solution helps protect sensitive data regarding privacy policies. Moreover, the authors developed an algorithm that detects the conflict of PA. It improves the rules of Enterprise Privacy Authorization Language (EPAL) [17].

For other PbD aspects, Bu et al. [18] created research questions about implementing privacy into a system and interviewed 253 engineers in the IT industry. The authors evaluated the answers to those research questions via statistical hypotheses. PriBioAuth [19] is a framework that tries to improve the privacy of biometric-based remote user authentication (BRUA). This paper proposed a biometric encryption protocol that encrypts sensitive data inside an anonymous session to unlink user identity.

As consent is one of the six principles described in the GDPR, which allows data subjects to control their data processing under a specific purpose, we consider consent management an essential component of system design for enabling PbD and GDPR. In this paper, we propose formal models for consent management and adopt the Event-B [20] method to describe and verify the models. Event-B is a formal method used to evaluate a system modeling to ensure that all states and transitions are reachable and work correctly. We also used Rodin [21] to develop the system models in Event-B. It helps produce proof obligations and attempts automatically using an automated theorem prover.

3. Background

We reviewed GDPR articles from a system design perspective to build GDPR-aware system models related to PbD [22]. The key roles in GDPR include the 1) data subject, 2) data controller, and 3) data processor. A data subject has full control of his/her data [Article 4(1) describes personal data as information that leads to the recognition of an individual]. The data controller is the organization or person who establishes policies for managing a life cycle of personal data processing, as described in Article 4(7). Finally, the data processor is the organization or person who manipulates individual data according to the policies given by the data controller, as described in Article 4(8). We then defined a set of primitive state machines that cover the basics of consent management functionality, consisting of four state machines: 1) the restricted processing state machine (RPSM), 2) the withdrawal approval state machine (WASM), 3) the portable approval state machine (PASM), and 4) the consent renewal state machine (CRSM). Moreover, we mapped each state machine to GDPR articles (Table 1), which helps developers better understand how to translate GDPR articles into system requirements and design.

To define a set of states and transitions in RPSM, we determined the logic involved in processing activities by following privacy methods included in Article 5. This article outlines the context of personal data processing that respects six data protection principles as follows: 1) it requires that personal data are stored and processed legitimately ('lawfulness,

fairness and transparency'), 2) the purpose of data processing must be clearly defined before beginning the process ('purpose limitation'), 3) personal data should only include a minimum amount of data that is strictly necessary to accomplish a specific purpose ('data minimization'), 4) personal data must be complete and kept up-to-date ('accuracy'), 5) the data controller must ensure that personal data will be only retained for a necessarily limited period ('storage limitation'), and 6) personal data must be ensured with consistency and confidentiality over its life cycle ('integrity and confidentiality'). The responsibilities of the data controller must comply with these fundamentals.

Furthermore, we built WASM as a model dealing with the right to withdraw consent. Article 7(3) describes that the data subjects are able at all times to revoke consent for the processing of their data. After revoking the consent, personal data should be erased automatically [23]. This revoking is also known as the right to erasure ('right to be forgotten') under Articles 17 and 19.

The right to data portability, GDPR Article 20, permits data subjects to control their data by receiving and transferring personal data in a machine-readable format across controllers. We modeled this discrete behavior through the state transitions in PASM.

For the renewal of consent effects within GDPR Article 6(1), the data controller may offer a data subject to extend the retention period to continue using the products and services. If the data subject accepts the retention offer, the data controller or the data processor can legitimately process his/her data. However, if the data subject declines the retention offer, the data controller must revoke the data subject's consent. We also modeled this discrete behavior through the state transitions in CRSM.

Except for consent, five more of the six legal bases are set out in Article 6 of the GDPR as follows: 1) the contract requires that data processing is objectively necessary to fulfill a contract with data subjects, and data controllers must determine which legal basis to rely upon for processing personal data, e.g., the obligations of payment services, 2) the legitimate interest requires that data processing is necessary for ways that data subjects usually expect, and organizations conduct their personal data to meet its purposes, e.g., fraud prevention, 3) the vital interest requires that data processing is necessary to save individuals' life, e.g., emergency medical care, 4) the legal requirement requires that data processing is necessary to fulfill a legal obligation, e.g., information security, and 5) the public interest requires that data processing is necessary for the performance of public functions that occur by public authorities, e.g., a public body's tasks. Therefore, the difference between consent and contract is that consent must be informed and freely given, while a contract requires that data processing is necessary to comply with the data controller's obligation.

Within our proposed, an Event-B is a modeling method in mathematical terms [24] for proving if the system satisfies the required properties. Event-B is composed of two parts: 1) the context, a static part of a model, which contains types and constants, and 2) the machine representing a dynamic part that comprises state variables, invariants, and events. Moreover, the Event-B model requires preserving the consistency with a set of proof obligations (POs) produced by each invariant rule.

The POs are discharged to ensure that actions in events are feasible under guards. The guards are pre-conditions that are necessary for an event to be executed. For each discrete event, when guards $G(v)$ are valid, the system state variables v will be modified by actions $S(v)$, as shown in (1).

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

The POs are guaranteed that each invariant is preserved by event, as shown in (2).

$$\begin{array}{l} I(V) \\ G(V) \\ \vdash \\ I(S(v)) \end{array} \tag{2}$$

Event-B is constructed through refinement-based methods. The refinement-based method is a technique for creating complex systems starting from a small portion of each feature, one at a time, and then gradually adding features. It makes a model simpler and easier to prove.

To develop Event-B models, we first need to install Java and Rodin Platform (i.e., an Eclipse IDE for Event-B) with instructions: http://www.event-b.org/install.html. Then we start Rodin. To create a new project, we click the File menu and then choose Event-B Project. Next, we create a new context and machine within the Event-B Component menu. After creating a new machine, we then select the new context extending. Moreover, when saving a context or machine, the auto-prover is automatically discharged on POs. Hence, the Rodin Platform is easier to learn and use.

To start with Event-B, we present some main elements used on these state machines. We then define 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. The notation examples 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 for specifying 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

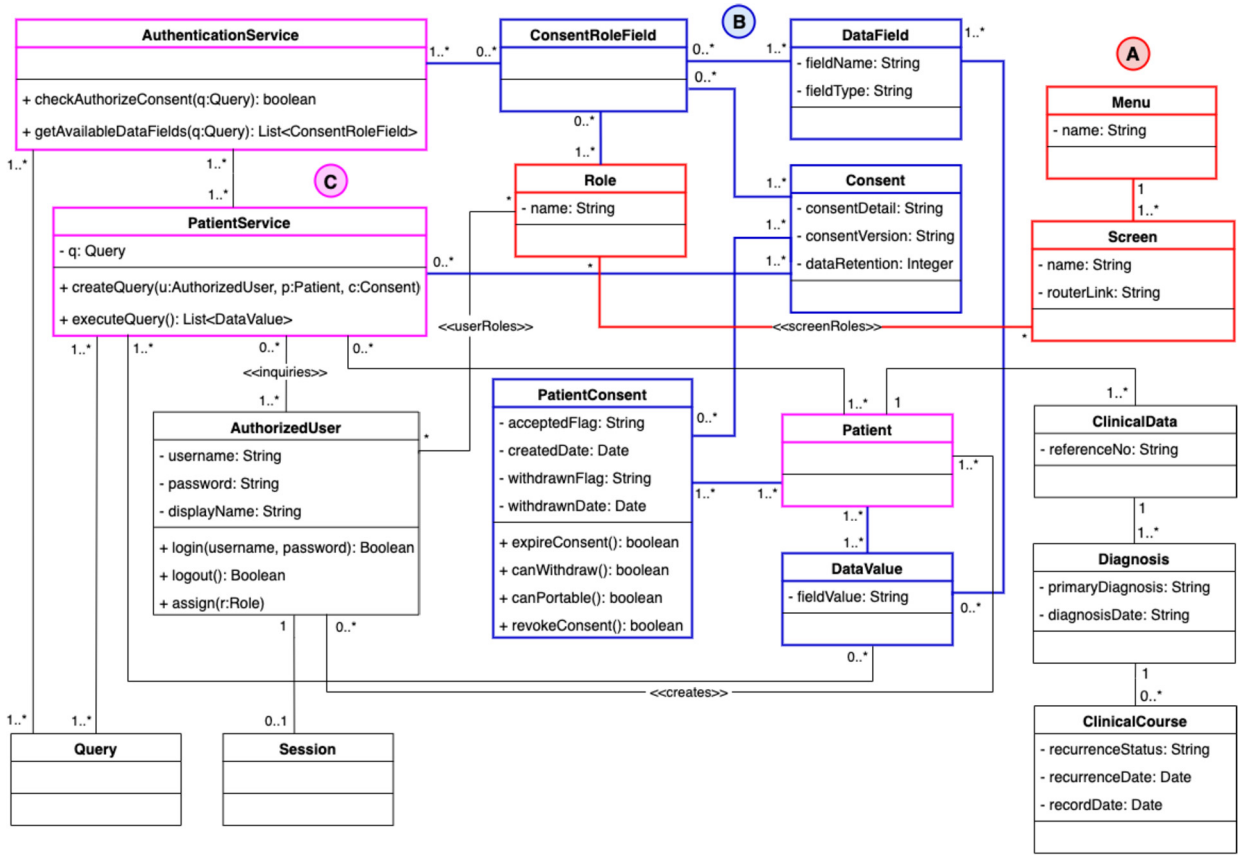


Fig. 1. Class diagram demonstrating how a software platform for cancer precision medicine manages roles and permissions to restrict users' access to screens. (A) an authentication module associated with users, roles, and screens. (B) new classes added to RUN-ONCO for supporting dynamic access attributes within role-based consent. (C) relevant classes needed to be enhanced to support consent management.

restriction ($S \lhd T$), domain subtraction ($S \lhd\!\!\!\!- T$), range restriction ($S \rhd T$), range subtraction ($S \rhd\!\!\!\!- T$), and relational image ($r[S]$). More information about Event-B notation can be found in [25].

4. Consent management state machines

This research proposes a set of formal models which integrate privacy concerns into software development under the GDPR. According to Article 4(11) GDPR, consent is a data subject's voluntary agreement to permit either a data controller or a data processor to process his/her personal data under specific conditions. We considered consent management as an essential component of the system design [26,27]. This means a system must not process personal data without the validity of a data subject's consent. In this paper, we built state machines to depict the dynamic behavior of privileged permissions based on the relationships of data subject's consents, user roles, and data subject's data fields.

Following PbD concepts and GDPR guidelines present in Table 1, demonstrated via a software platform for cancer precision medicine called RUN-ONCO [28]. RUN-ONCO allows users (i.e., oncologists, nurses, researchers) to manage and analyze patient clinical and genetic data, which assists oncologists in designing treatment plans for patients with cancers. Patients need to sign consent before an authorized user enters their clinical and genetic data into the platform. Fig. 1A shows how RUN-ONCO supports authentication based on roles but lacks the consent management functionality. The informed consent process for clinical trials has been paper-based and outside the platform. Without built-in consent management functionality, a platform is difficult to control and maintain patients' privacy preferences. To implement a consent management functionality for an existing system without clear guidelines, developers will need to spend much time analyzing and redesigning the system without knowing if the redesigned platform covers GDPR requirements. To enhance RUN-ONCO support consent management (Figs. 1B and 1C), by following RPSM, we first need to alter the *Patient* class structure to support dynamic access attributes within role-based consent. Second, we further create the *PatientConsent* class to hold patients' consent. To manage the right to withdraw consent (WASM), the right to data portability (PASM), and consent renewal (CRSM), we then update the *PatientConsent* class by adding methods that obtain the logic of the following state machines. Third, we must modify logic in the *AuthenticationService* class to manage the authorized access patients' attributes within

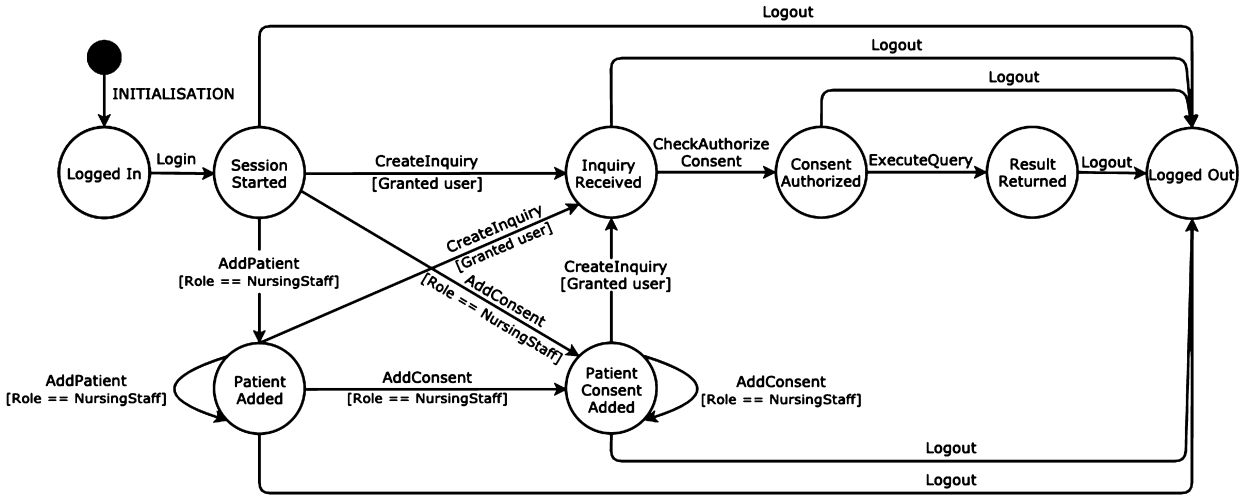


Fig. 2. Restricted Processing State Machine (RPSM) describing the transition states and events used to restrict the processing of personal data.

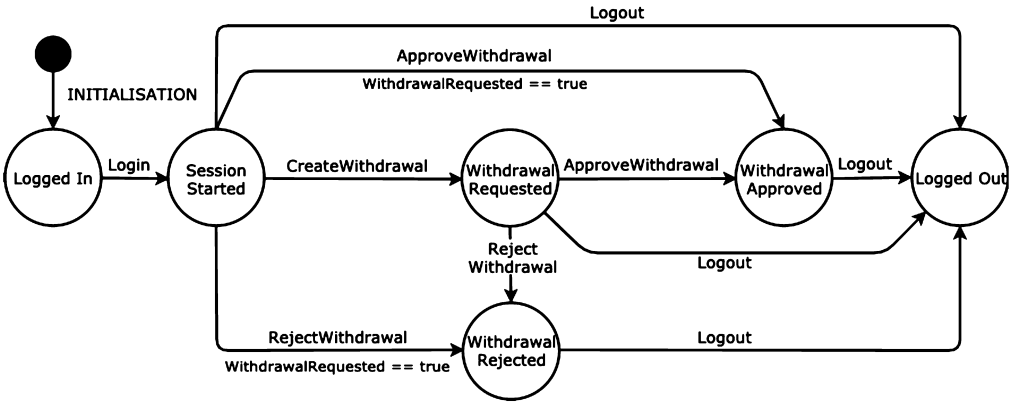


Fig. 3. Withdrawal Approval State Machine (WASM) describing the transition states and events used to manage a consent revocation request.

role-based consent. Fourth, the *PatientService* class needs to modify the logic for restricting patient information retrieval according to given authorization.

We provided four state machines that cover the main aspects of consent management. First, the RPSM explains the behavior of restricting unauthorized user access from storing and processing personal data (Fig. 2). Based on RPSM, a user must first login to access the platform. By logging in, the user with the NursingStaff role will be able to add a new patient and informed consent. Moreover, to access the patient's personal data, a user has been granted a role based on the patient's consent. Second, WASM explains the behavior of approval for withdrawing a data subject's consent and deleting his/her personal data (Fig. 3). Based on WASM after a patient requests to withdraw consent, the user with the LegalStaff role login to the platform and initiates a withdrawal process. A user with the LegalApprover role will then review a withdrawal request based on the initiated process. The platform allows patients to withdraw consent at any time, as long as the patient has the adequate capacity to make decisions about medical treatment. After assessing a patient's capacity, if a patient can make his/her own treatment decisions, the approver will approve to revoke consent and submit a delete request to erase the patient's personal data. Otherwise, the approver will reject the withdrawal request.

Third, PASM explains approval behavior for transferring a data subject's personal data (Fig. 4). Based on PASM, after a patient requests a portable copy of the personal data, the user with the LegalStaff role login to the platform and initiates a portable process. The platform offers data portability that allows patients to request all relevant health and genetic data, as long as the patient accepts prerequisite conditions (e.g., a fee for preparing and transmitting personal data to other data controllers). The approver will approve the portable request if the patient accepts prerequisite conditions. Otherwise, the request will be rejected. Fourth, CRSM explains approval behavior for extending the retention period of a data subject's consent (Fig. 5). Based on CRSM, the user with the LegalStaff role login to the platform and initiates a renewal process. The patient will then review a renewal request based on the initiated process. The platform offers a mechanism that allows patients to increase the retention period for keeping the personal data it collects and processes. After the patient replies accept status (i.e., approve, reject) to the platform, the legal staff responds, followed by accept status. If the patient approves,

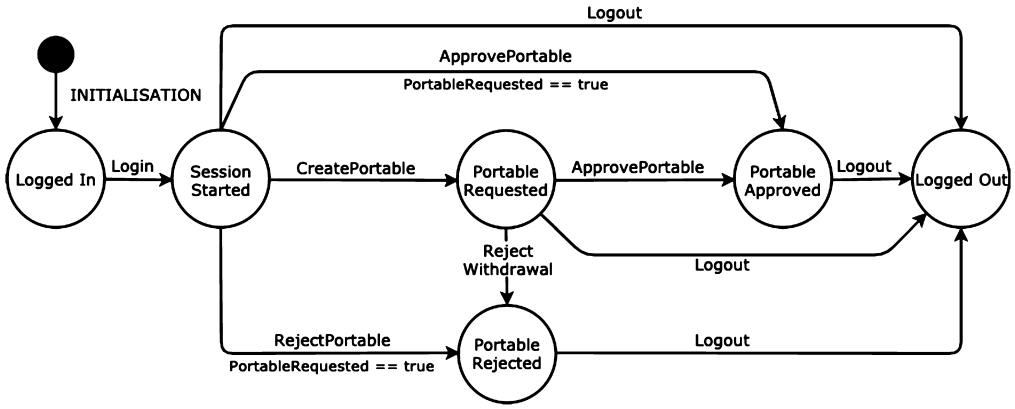


Fig. 4. Portable Approval State Machine (PASM) describing the transition states and events used to manage a data transferring request.

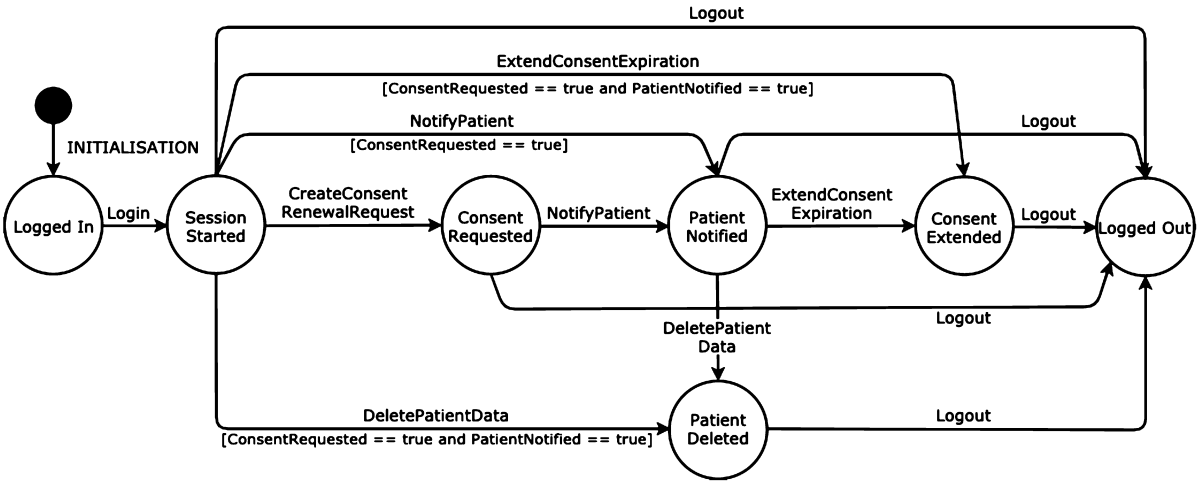


Fig. 5. Consent Renewal State Machine (CRSM) describing the transition states and events used to manage a data retention request.

the legal staff increases the retention period within informed consent. Otherwise, the legal staff submits a delete request to erase the patient's personal data.

5. Formal development in Event-B

We created an Event-B context and defined necessary sets, constants, and axioms that are relevant to health information privacy as follows: 1) PATIENTS is a set of data subjects, 2) SESSIONS represents a set of sessions associated with an authorized user (i.e., AUTHORIZED_USERS), 3) ROLES (e.g., NursingStaff, Oncologist, LabStaff) specifies a set of user permissions to prevent unauthorized access attempts, 4) FIELDS is a set of patient data fields (e.g., HN, Name, Age, Gender), and 5) STATUSES is a set of workflow statuses (e.g., Void, Approved, Rejected). The state machines will refer to this context, which contains global static variables to construct the states and transitions. We built the state machines and defined preserved invariants as the properties of the states using common naming, e.g., inv1, inv2. Events represent state transitions in Event-B. For each event, we defined guards as preconditions and actions as state variable assignments using the common naming, e.g., grd1, grd2, and act1, act2, respectively.

5.1. Restricted processing state machine (RPSM)

The RPSM (Fig. 2), created based on the Event-B method, describes the dynamic behavior of restricted data processing in terms of events. For this state machine, we defined invariants that hold all possible states as follows:

inv1: $sessions \in SESSIONS \rightarrowtail AUTHORIZED\_USERS$

inv2: $userRoles \in AUTHORIZED\_USERS \leftrightarrow ROLES$

inv3: $pc \in PATIENTS \leftrightarrow CONSENTS$

inv4: $patients \in \mathbb{P}(PATIENTS)$

inv5: $crf \in CONSENTS \pfun (ROLES \rel FIELDS)$

inv6: $queries \in AUTHORIZED\_USERS \pfun (QUERIES \rel PATIENTS)$

inv7: $pf \in AUTHORIZED\_USERS \pfun (PATIENTS \rel FIELDS)$

inv8: $authorizedConsent \in AUTHORIZED\_USERS \pfun (PATIENTS \rel CONSENTS)$

The variable *sessions* holds the one-to-one relationship between SESSIONS and AUTHORIZED_USERS, which means a single session can contain only one user. To limit the data breach risk, we applied role-based access control (RBAC) in the model and defined the userRoles as a relationship between AUTHORIZED_USERS and ROLES. It indicates that each user can have multiple roles. The variable *patients* contains the set of PATIENTS during the refinement process. According to GDPR, we need a patient's consent to process data. Hence, we declared the *pc* as a set of ordered pairs ($p \mapsto c$) where $p \in$ PATIENTS and $c \in$ CONSENTS. The use of *pc* here specifies that a patient can have more than one consent. The *crf* defines ($c \mapsto rf$) as a set of ordered pairs where $c \in$ CONSENTS, $rf \in$ ROLES $\leftrightarrow$ FIELDS, which combines the relationships of consents, roles, and data fields to restrict user's access over the specific fields of data based on the given consent of data subjects. The model allows a data controller or a data processor to execute a query per data subject to minimize the risk of retrieving large amounts of personal data by creating the variable named *queries*. The *queries* defines ($u \mapsto qp$) as a set of ordered pair where $u \in$ AUTHORIZED_USERS, $qp \in$ QUERIES $\leftrightarrow$ PATIENTS to hold personal data inquiries. We stored the result of a query in variable *pf*, which is a set of ordered pairs ($u \mapsto pf$) where $u \in$ AUTHORIZED_USERS and $pf \in$ PATIENTS $\leftrightarrow$ FIELDS. The *pf* represents the final output of RPSM that describes how the model provides consent-based permission for each user to perform on specified data fields. We defined *authorizedConsent* ($u \mapsto pc$) as a set of ordered pairs where $u \in$ AUTHORIZED_USERS and $pc \in$ PATIENTS $\leftrightarrow$ CONSENTS, indicating the valid consent for the authorized user.

The INTIALISATION is an event that was fired first. It allows the initialization of arbitrary values and establishes invariants before other events are executed. Listing 1 introduces the Login event. The guards are defined with three preconditions. First, the guard grd1 ensures that any session *s* is a member of SESSIONS and *s* does not exist in the domain of *sessions*. Second, the guard grd2 ensures that any user *u* is a member of AUTHORIZED_USERS and *u* does not exist in the range of *sessions*. Third, the guard grd3 ensures that adding an ordered pair ($s \mapsto u$) into *sessions* must satisfy the invariant inv1. Whenever all guards of the Login event are valid, the action act1 adds an ordered pair ($s \mapsto u$) to the *sessions*, which indicates that user has successfully logged in.

Listing 2 shows how we formally modeled the adding of a new patient using Event-B. The guards are defined with four preconditions. First, the guard grd1 ensures that the user successfully got the session and the user role is within the domain *userRoles*. Second, the guard grd2 ensures that one of the user roles is a nursing staff. Third, the guard grd3 ensures that the patient does not exist in the variable *patients*. Fourth, the guard grd4 ensures that the AddPatient event does not fire after entering the inquiry states. Whenever all guards are valid, the action act1 adds the patient *p* to the *patients*.

Listing 3 shows the formal model of how a new patient's consent is added to the system. The guards are defined with six preconditions. First, the guard grd1 ensures that the user is successfully logged in with the user role known by the system. Second, the guard grd2 ensures that one of the user roles is a nursing staff. Third, the guard grd3 ensures that any patient *p* is a member of *patients* and consent *c* is a member of the domain *crf*. Fourth, the guard grd4 ensures that a new ordered pair ($p \mapsto c$) does not exist in the *pc*. Fifth, the guard grd5 ensures that adding an ordered pair ($p \mapsto c$) into variable *pc* must satisfy the invariant inv3. Sixth, the guard grd6 ensures that the AddConsent event does not fire after entering the inquiry states. Whenever all guards are valid, the action act1 adds an ordered pair ($p \mapsto c$) to the *pc*.

Listings 4–6 show how we formally model the handling of a user inquiry, starting from creating an inquiry (Listing 4), verifying the consent validation (Listing 5), and executing the inquiry (Listing 6). The CreateInquiry event (Listing 4) is used to prepare a new query under the currently logged on user. The guards are defined with three preconditions. First, the guard grd1 ensures that the user is successfully logged in with the user role known by the system. Second, the guard grd2 ensures that any query *q* is a member of QUERIES, patient *p* is a member of the domain *pc*, and *session(s)* does not exist

Login $\widehat{=}$

Any s, u **Where**

$grd1 : s \in SESSIONS \land s \notin dom(sessions)$

$grd2 : u \in AUTHORIZED\_USERS \land u \notin ran(sessions)$

$grd3 : sessions \cup \{s \mapsto u\} \in SESSIONS \pinj AUTHORIZED\_USERS$

Then

$act1 : sessions := sessions \cup \{s \mapsto u\}$

End

Listing 1. The Login event.

AddPatient $\widehat{=}$

Any s, p **Where**

$grd1 : s \in dom(sessions) \land sessions(s) \in dom(userRoles)$

$grd2 : \exists r \cdot r \in userRoles[sessions[\{s\}]] \land r = NursingStaff$

$grd3 : p \in PATIENTS \land p \notin patients$

$grd4 : sessions(s) \notin dom(queries)$

Then

$act1 : patients := patients \cup \{p\}$

End

Listing 2. The AddPatient event.

AddConsent $\widehat{=}$

Any s, p, c **Where**

$grd1 : s \in dom(sessions) \land sessions(s) \in dom(userRoles)$

$grd2 : \exists r \cdot r \in userRoles[sessions[\{s\}]] \land r = NursingStaff$

$grd3 : p \in patients \land c \in dom(crf)$

$grd4 : p \mapsto c \notin pc$

$grd5 : pc \cup \{p \mapsto c\} \in PATIENTS \leftrightarrow CONSENTS$

$grd6 : sessions(s) \notin dom(queries)$

Then

$act1 : pc := pc \cup \{p \mapsto c\}$

End

Listing 3. The AddConsent event.

CreateInquiry $\widehat{=}$

Any s, q, p **Where**

$grd1 : s \in dom(sessions) \land sessions(s) \in dom(userRoles)$

$grd2 : q \in QUERIES \land p \in dom(pc) \land sessions(s) \notin dom(queries)$

$grd3 : queries \lhd\!\!+ \{sessions(s) \mapsto \{q \mapsto p\}\} \in$

$AUTHORIZED\_USERS \nrightarrow (QUERIES \leftrightarrow PATIENTS)$

Then

$act1 : queries(sessions(s)) := \{q \mapsto p\}$

End

Listing 4. The CreateInquiry event.

in the domain *queries*. Third, the guard grd3 ensures that when adding an ordered pair (q $\mapsto$ p) to the *queries(sessions(s))*, the invariant inv6 must be satisfied. Whenever all guards are valid, the action act1 adds an ordered pair (q $\mapsto$ p) to the *queries(sessions(s))*.

The CheckAuthorizeConsent event (Listing 5) is used to verify if the patient's consent does not expire. The guards are defined with six preconditions. First, the guard grd1 ensures that the user is successfully logged in and the user has created queries. Second, the guard grd2 ensures that *consentExpired* is a member of the boolean and *consentExpired* is FALSE. Third, the guard grd3 ensures that the consent *c* is a member of *pc[{p}]* and *c* is a member of the domain *crf*. Fourth, the guard grd4 ensures that one of the user roles of the logged on user is a member of the domain *crf(c)*. Fifth, the guard grd5 ensures that a new ordered pair (p $\mapsto$ c) does not exist in the domain *authorizedConsent*. Sixth, the guard grd6 ensures that when adding an ordered pair (p $\mapsto$ c) to the *authorizedConsent(sessions(s))*, the invariant inv8 must be satisfied. Whenever all guards are valid, the action act1 adds an ordered pair (p $\mapsto$ c) to the *authorizedConsent(sessions(s))*.

CheckAuthorizeConsent $\widehat{=}$

Any s, p, c, $consentExpired$ **Where**

$grd1 : s \in dom(sessions) \land sessions(s) \in dom(queries)$

$grd2 : consentExpired \in BOOL \land consentExpired = FALSE$

$grd3 : c \in pc[\{p\}] \land c \in dom(crf)$

$grd4 : \exists r \cdot r \in userRoles[sessions[\{s\}]] \land r \in dom(crf(c))$

$grd5 : sessions(s) \notin dom(authorizedConsent)$

$grd6 : authorizedConsent \lhd\!\!+ \{sessions(s) \mapsto \{p \mapsto c\}\} \in$

$AUTHORIZED\_USERS \pfun (PATIENTS \rel CONSENTS)$

Then

$act1 : authorizedConsent(sessions(s)) := \{p \mapsto c\}$

End

Listing 5. The CheckAuthorizeConsent event.

ExecuteQuery $\widehat{=}$

Any s, p, c **Where**

$grd1 : s \in dom(sessions) \land sessions(s) \in dom(queries)$

$grd2 : p \in ran(queries(sessions(s))) \land c \in dom(crf)$

$grd3 : sessions(s) \in dom(authorizedConsent) \land p \mapsto c \in$

$authorizedConsent(sessions(s))$

$grd4 : sessions(s) \notin dom(pf)$

$grd5 : pf \lhd\!\!+ \{sessions(s) \mapsto \{p\} \times ran(userRoles[sessions[\{s\}]] \lhd crf(c))\} \in$

$AUTHORIZED\_USERS \pfun (PATIENTS \rel FIELDS)$

Then

$act1 : pf(sessions(s)) := \{p\} \times ran(userRoles[sessions[\{s\}]] \lhd crf(c))$

End

Listing 6. The ExecuteQuery event.

The ExecuteQuery event (Listing 6) is used to get the result of a query. The guards are defined with five preconditions. First, the guard grd1 ensures that the user is successfully logged in and the user has created queries. Second, the guard grd2 ensures that any patient p is a member of the range of *queries(sessions(s))* and c is a member of the domain *crf*. Third, the guard grd3 ensures that *sessions(s)* is a member of the domain *authorizedConsent* and an ordered pair (p $\mapsto$ c) is a member of *authorizedConsent(sessions(s)*. Fourth, the guard grd4 ensures that *sessions(s)* does not exist in a domain *pf*. The grd4 represents that the query has not yet been executed within the user session. Fifth, the guard grd5 ensures that when adding a cartesian product *{p}* $\times$ *ran(userRoles[sessions[{s}]]* $\lhd$ *crf(c))* to the *pf(sessions(s))*, the invariant inv7 must be satisfied. The variable *pf* represents the result of the query based on consent-permission which is defined in the variable *crf*. Whenever all guards are valid, the action act1 adds a cartesian product *{p}* $\times$ *ran(userRoles[sessions[{s}]]* $\lhd$ *crf(c))* to the *pf(sessions(s))*.

The Logout event (Listing 7) is fired when a user signs out of the system. The guards are defined with five preconditions. First, the guard grd1 ensures that the user is successfully logged in. Second, the guard grd2 ensures that removing *sessions(s)* from *queries* must satisfy the invariant inv6. Third, the guard grd3 ensures that removing *sessions(s)* from *authorizedConsent* must satisfy the invariant inv8. Fourth, the guard grd4 ensures that removing *sessions(s)* from *pf* must satisfy the invariant inv7. Fifth, the guard grd5 ensures that removing *sessions(s)* from *sessions* must satisfy the invariant inv1. Whenever all guards of the Logout event are valid, the action act1 removes *sessions(s)* from *queries*, action act2 removes *sessions(s)* from *authorizedConsent*, action act3 removes *sessions(s)* from *pf*, and action act4 removes *sessions(s)* from *sessions*.

Logout $\widehat{=}$

Any s **Where**

$grd1 : s \in dom(sessions)$

$grd2 : \{sessions(s)\} \lhd queries \in$

$AUTHORIZED\_USERS \pfun (QUERIES \rel PATIENTS)$

$grd3 : \{sessions(s)\} \lhd authorizedConsent \in$

$AUTHORIZED\_USERS \pfun (PATIENTS \rel CONSENTS)$

$grd4 : \{sessions(s)\} \lhd pf \in AUTHORIZED\_USERS \pfun (PATIENTS \rel FIELDS)$

$grd5 : sessions \rhd \{sessions(s)\} \in SESSIONS \tinj AUTHORIZED\_USERS$

Then

$act1 : queries := \{sessions(s)\} \lhd queries$

$act2 : authorizedConsent := \{sessions(s)\} \lhd authorizedConsent$

$act3 : pf := \{sessions(s)\} \lhd pf$

$act4 : sessions := sessions \rhd \{sessions(s)\}$

End

Listing 7. The Logout event.

5.2. Withdrawal approval state machine (WASM)

The WASM (Fig. 3) was created based on the Event-B method to describe the dynamic behavior of the model for revoking an individual consent and automatically deleting personal data. We defined the invariants for the WASM model as follows. The first three invariants are the same as of RPSM.

inv1: $sessions \in SESSIONS \tinj AUTHORIZED\_USERS$

inv2: $userRoles \in AUTHORIZED\_USERS \rel ROLES$

inv3: $pc \in PATIENTS \rel CONSENTS$

inv4: $withdrawalState \in (PATIENTS \rel CONSENTS) \tinj STATUSES$

inv5: $markAsDeleted \in PATIENTS \rel CONSENTS$

Additionally, we declared two more variables in the context to support the refinement of WASM. First, the *withdrawalState* defines (pc $\mapsto$ status) as a set of ordered pairs, where pc $\in$ PATIENTS $\leftrightarrow$ CONSENTS, and status $\in$ STATUSES that holds the status of the withdrawal request. Second, the *markAsDeleted* contains the relationship between PATIENTS and CONSENTS that represents the patient as deleted under the consent.

The INTIALISATION event gets fired first to initialize the variables. Then the Login event starts to get a new session which holds a user role. The CreateWithdrawal event (Listing 8) is used to initiate a withdrawal request. The guards are defined with four preconditions. First, the guard grd1 ensures that the user successfully got the session and the user role is within the domain *userRoles*. Second, the guard grd2 ensures that one of the user roles is a legal staff. Third, the guard grd3 ensures that any patient p is a member of the domain *pc*, where consent c is a member of the range *pc*, and the ordered pair (p $\mapsto$ c) does not exist in the domain *withdrawalState*. Fourth, the guard grd4 ensures that when adding Void status to the *withdrawalState({p* $\mapsto$ *c})*, the invariant inv4 must be satisfied. Whenever all guards are valid, the action act1 adds a status Void to the *withdrawalState({p* $\mapsto$ *c})*, which will trigger the approval workflow.

Listing 9 shows the formal model of how to approve the consent withdrawal. The guards are defined with six preconditions. First, the guard grd1 ensures that the user successfully got the session and the user role is within the domain *userRoles*. Second, the guard grd2 ensures that one of the user roles is a legal approver. Third, the guard grd3 ensures that *pc1* is a member of the domain *withdrawalState* and the status of the *withdrawalState(pc1)* is Void. Fourth, the guard grd4 ensures that when updating Void to Approved status must satisfy the invariant inv4. Fifth, the guard grd5 ensures that *canWithdraw* is a member of a boolean and *canWithdraw* is TRUE. The TRUE boolean here indicates that all required activities before withdrawal were done. Sixth, the guard grd6 ensures that when adding *pc1* to the *markAsDeleted*, the invariant inv5 must be satisfied. Whenever all guards are valid, the action act1 updates the *withdrawalState({p* $\mapsto$ *c})* from Void to Approved status, and act2 adds *pc1* to *markAsDeleted*.

Otherwise, the RejectWithdrawal event (Listing 10) will be fired if the variable *canWithdraw* is FALSE, assuming that some required activities were not completed. The status of *withdrawalState(pc1)* will then be changed from Void to Rejected

CreateWithdrawal $\widehat{=}$

Any s, p, c **Where**

$grd1 : s \in dom(sessions) \wedge sessions(s) \in dom(userRoles)$

$grd2 : \exists r \cdot r \in userRoles[sessions[\{s\}]] \wedge r = LegalStaff$

$grd3 : p \in dom(pc) \wedge c \in ran(pc) \wedge \{p \mapsto c\} \notin dom(withdrawalState)$

$grd4 : withdrawalState \lhd\!\!\!- \{\{p \mapsto c\} \mapsto Void\} \in$

$(PATIENTS \leftrightarrow CONSENTS) \rightarrowtail\!\!\!\!\!\!+ STATUSES$

Then

$act1 : withdrawalState(\{p \mapsto c\}) := Void$

End

Listing 8. The CreateWithdrawal event.

ApproveWithdrawal $\widehat{=}$

Any $s, pc1, canWithdraw$ **Where**

$grd1 : s \in dom(sessions) \wedge sessions(s) \in dom(userRoles)$

$grd2 : \exists r \cdot r \in userRoles[sessions[\{s\}]] \wedge r = LegalApprover$

$grd3 : pc1 \in dom(withdrawalState) \wedge withdrawalState(pc1) = Void$

$grd4 : withdrawalState \lhd\!\!\!- \{pc1 \mapsto Approved\} \in$

$(PATIENTS \leftrightarrow CONSENTS) \rightarrowtail\!\!\!\!\!\!+ STATUSES$

$grd5 : canWithdraw \in BOOL \wedge canWithdraw = TRUE$

$grd6 : markAsDeleted \lhd\!\!\!- pc1 \in PATIENTS \leftrightarrow CONSENTS$

Then

$act1 : withdrawalState(pc1) := Approved$

$act2 : markAsDeleted := markAsDeleted \lhd\!\!\!- pc1$

End

Listing 9. The ApproveWithdrawal event.

RejectWithdrawal $\widehat{=}$

Any $s, pc1, canWithdraw$ **Where**

$grd1 : s \in dom(sessions) \wedge sessions(s) \in dom(userRoles)$

$grd2 : \exists r \cdot r \in userRoles[sessions[\{s\}]] \wedge r = LegalApprover$

$grd3 : pc1 \in dom(withdrawalState) \wedge withdrawalState(pc1) = Void$

$grd4 : withdrawalState \lhd\!\!\!- \{pc1 \mapsto Rejected\} \in$

$(PATIENTS \leftrightarrow CONSENTS) \rightarrowtail\!\!\!\!\!\!+ STATUSES$

$grd5 : canWithdraw \in BOOL \wedge canWithdraw = FALSE$

Then

$act1 : withdrawalState(pc1) := Rejected$

End

Listing 10. The RejectWithdrawal event.

according to the action act1. In both cases, the request must be approved or rejected by the legal approver. Especially in the ApproveWithdrawal event, we defined the *markAsDeleted* to hold the deleted patients for the approved cases. The Logout event is fired to indicate that the user is no longer in the system.

CreatePortable $\widehat{=}$

Any s, p, c **Where**

$grd1 : s \in dom(sessions) \wedge sessions(s) \in dom(userRoles)$

$grd2 : \exists r \cdot r \in userRoles[sessions[\{s\}]] \wedge r = LegalStaff$

$grd3 : p \in dom(pc) \wedge c \in ran(pc) \wedge \{p \mapsto c\} \notin dom(portableState)$

$grd4 : portableState \lhd\!\!- \{\{p \mapsto c\} \mapsto Void\} \in$

$(PATIENTS \leftrightarrow CONSENTS) \rightarrowtail\!\!\!\!+ STATUSES$

Then

$act1 : portableState(\{p \mapsto c\}) := Void$

End

Listing 11. The CreatePortable event.

ApprovePortable $\widehat{=}$

Any $s, pc1, canPortable$ **Where**

$grd1 : s \in dom(sessions) \wedge sessions(s) \in dom(userRoles)$

$grd2 : \exists r \cdot r \in userRoles[sessions[\{s\}]] \wedge r = LegalApprover$

$grd3 : pc1 \in dom(portableState) \wedge portableState(pc1) = Void$

$grd4 : portableState \lhd\!\!- \{pc1 \mapsto Approved\} \in$

$(PATIENTS \leftrightarrow CONSENTS) \rightarrowtail\!\!\!\!+ STATUSES$

$grd5 : canPortable \in BOOL \wedge canPortable = TRUE$

Then

$act1 : portableState(pc1) := Approved$

End

Listing 12. The ApprovePortable event.

5.3. Portable approval state machine (PASM)

The PASM (Fig. 4) created based on Event-B describes the dynamic behavior of the model allowing patients to port their personal data. The first three invariants of the model are the same as the previous two models and a new variable named *portableState* was introduced to hold the status of data portability request.

inv1: $sessions \in SESSIONS \rightarrowtail\!\!\!\!+ AUTHORIZED\_USERS$

inv2: $userRoles \in AUTHORIZED\_USERS \leftrightarrow ROLES$

inv3: $pc \in PATIENTS \leftrightarrow CONSENTS$

inv4: $portableState \in (PATIENTS \leftrightarrow CONSENTS) \rightarrowtail\!\!\!\!+ STATUSES$

The behavior of PASM is similar to the WASM but is used for different purposes. After initializing the variables and creating a new session, the CreatePortable event (Listing 11) will be started. The guards are defined with four preconditions. First, the guard grd1 ensures that the user successfully got the session and the user role is within the domain *userRoles*. Second, the guard grd2 ensures that one of the user roles is a legal staff. Third, the guard grd3 ensures that any patient *p* is a member of the domain *pc*, consent *c* is a member of the range *pc*, and the new ordered pair (p $\mapsto$ c) does not exist in the domain *portableState*. Fourth, the guard grd4 ensures that when adding Void status to the *portableState({p* $\mapsto$ *c})*, the invariant inv4 must be satisfied. Whenever all guards are valid, the action act1 adds the status Void to the *portableState({p* $\mapsto$ *c})*.

After the CreatePortable event is done, the ApprovePortable event (Listing 12) will be fired if the variable *canPortable* is TRUE. The status of *portableState(pc1)* will then be changed from Void to Approved according to the action act1. Otherwise, the RejectPortable event (Listing 13) will be fired to change the status from Void to Rejected. In both cases, the portability request must be determined by the legal approver.

RejectPortable $\widehat{=}$

Any $s, pc1, canPortable$ **Where**

$grd1 : s \in dom(sessions) \wedge sessions(s) \in dom(userRoles)$

$grd2 : \exists r \cdot r \in userRoles[sessions[\{s\}]] \wedge r = LegalApprover$

$grd3 : pc1 \in dom(portableState) \wedge portableState(pc1) = Void$

$grd4 : portableState \Leftarrow \{pc1 \mapsto Rejected\} \in$

$(PATIENTS \leftrightarrow CONSENTS) \rightarrowtail\!\!\!\!\!+ STATUSES$

$grd5 : canPortable \in BOOL \wedge canPortable = FALSE$

Then

$act1 : portableState(pc1) := Rejected$

End

Listing 13. The RejectPortable event.

5.4. Consent renewal state machine (CRSM)

The CRSM model (Fig. 5) created by Event-B describes the dynamic behavior of the model to extend the renewal period of a consent. The first three invariants of the model are the same as the previous three models. We also defined four more invariants and variables to cover the refinement of CRSM as follows.

inv1: $sessions \in SESSIONS \rightarrowtail\!\!\!\!\!+ AUTHORIZED\_USERS$

inv2: $userRoles \in AUTHORIZED\_USERS \leftrightarrow ROLES$

inv3: $pc \in PATIENTS \leftrightarrow CONSENTS$

inv4: $isConsentExpired \in (PATIENTS \leftrightarrow CONSENTS) \rightarrowtail\!\!\!\!\!+ BOOL$

inv5: $markAsDeleted \in PATIENTS \leftrightarrow CONSENTS$

inv6: $markAsReceived \in PATIENTS \leftrightarrow CONSENTS$

inv7: $consentRenewalState \in (PATIENTS \leftrightarrow CONSENTS) \rightarrowtail\!\!\!\!\!+ STATUSES$

The first variable *isConsentExpired* is a set of ordered pairs represents by one-to-one relationship (pc $\mapsto$ expired) where $pc \in$ PATIENTS $\leftrightarrow$ CONSENTS, and expired $\in$ BOOL (i.e., TRUE or FALSE). The second variable *markAsDeleted* contains the relationship between PATIENTS and CONSENTS that represents the patient as deleted under the consent. The third variable *markAsReceived* contains the relationship between PATIENTS and CONSENTS that keeps track of the patient's incoming response to the renewal request. The fourth variable is *consentRenewalState*, which has held the status of consent renewal. It is a set of ordered pairs (pc $\mapsto$ status), where pc $\in$ PATIENTS $\leftrightarrow$ CONSENTS, and status $\in$ STATUSES that holds the status of patient's consent.

By default, the INTIALISATION event is fired to initialize the variables before executing a renewal request. The Login event is triggered to retrieve the user login information, and the session has started. The CreateConsentRenewalRequest event (Listing 14) is used to initiate a consent renewal request. The guards are defined with seven preconditions. First, the guard grd1 ensures that the user successfully got the session and the user role is within the domain *userRoles*. Second, the guard grd2 ensures that one of the user roles is a legal staff. Third, the guard grd3 ensures that any patient *p* is a member of the domain *pc*, consent *c* is a member of the range *pc*, and a new ordered pair (p $\mapsto$ c) does not exist in the domain *consentRenewalState*. Fourth, the guard grd4 ensures that *expired* is a member of a boolean and *expired* is TRUE. Fifth, the guard grd5 ensures that *isWithdrawn* is a member of a boolean and *isWithdrawn* is FALSE. Sixth, the guard grd6 ensures that when adding Void status to the *consentRenewalState({p* $\mapsto$ *c})*, the invariant inv7 must be still satisfied. Seventh, the guard grd7 ensures that when adding TRUE to the *isConsentExpired({p* $\mapsto$ *c})*, the invariant inv4 must be satisfied. Whenever all guards are valid, the action act1 adds a status Void to the *consentRenewalState({p* $\mapsto$ *c})*, and act2 adds TRUE to the *isConsentExpired({p* $\mapsto$ *c})*.

The NotifyPatient event (Listing 15) is used to notify the patient about extending the time period of consent. The guards are defined with five preconditions. First, the guard grd1 ensures that the user successfully got the session and the user role is within the domain *userRoles*. Second, the guard grd2 ensures that one of the user roles is a legal staff. Third, the guard grd3 ensures that *pc1* is not a subset of *markAsReceived*, *pc1* is a member of the domain *consentRenewalState*, and *consentRenewalState(pc1)* is equal to Void. Fourth, the guard grd4 ensures that the acceptStatus is a member of STATUSES but excludes Void. Fifth, the guard grd5 ensures that when updating the *acceptStatus* to the *consentRenewalState(pc1)*, the invariant inv7

CreateConsentRenewalRequest $\widehat{=}$

Any $s, p, c, expired, isWithdrawn$ **Where**

$grd1 : s \in dom(sessions) \wedge sessions(s) \in dom(userRoles)$

$grd2 : \exists r \cdot r \in userRoles[sessions[\{s\}]] \wedge r = LegalStaff$

$grd3 : p \in dom(pc) \wedge c \in ran(pc) \wedge \{p \mapsto c\} \notin dom(consentRenewalState)$

$grd4 : expired \in BOOL \wedge expired = TRUE$

$grd5 : isWithdrawn \in BOOL \wedge isWithdrawn = FALSE$

$grd6 : consentRenewalState \lhd\!\!- \{\{p \mapsto c\} \mapsto Void\} \in$

$(PATIENTS \leftrightarrow CONSENTS) \rightarrowtail\!\!\!\!\!\!+\;\; STATUSES$

$grd7 : isConsentExpired \lhd\!\!- \{\{p \mapsto c\} \mapsto TRUE\} \in$

$(PATIENTS \leftrightarrow CONSENTS) \rightarrowtail\!\!\!\!\!\!+\;\; BOOL$

Then

$act1 : consentRenewalState(\{p \mapsto c\}) := Void$

$act2 : isConsentExpired(\{p \mapsto c\}) := TRUE$

End

Listing 14. The CreateConsentRenewalRequest event.

NotifyPatient $\widehat{=}$

Any $s, pc1, acceptStatus$ **Where**

$grd1 : s \in dom(sessions) \wedge sessions(s) \in dom(userRoles)$

$grd2 : \exists r \cdot r \in userRoles[sessions[\{s\}]] \wedge r = LegalStaff$

$grd3 : pc1 \not\subseteq markAsReceived \wedge pc1 \in dom(consentRenewalState) \wedge$

$consentRenewalState(pc1) = Void$

$grd4 : acceptStatus \in STATUSES \setminus \{Void\}$

$grd5 : consentRenewalState \lhd\!\!- \{pc1 \mapsto acceptStatus\} \in$

$(PATIENTS \leftrightarrow CONSENTS) \rightarrowtail\!\!\!\!\!\!+\;\; STATUSES$

Then

$act1 : consentRenewalState(pc1) := acceptStatus$

$act2 : markAsReceived := markAsReceived \cup pc1$

End

Listing 15. The NotifyPatient event.

must be satisfied. Whenever all guards are valid, the action act1 adds the *acceptStatus* to the *consentRenewalState(pc1)*, and act2 adds pc1 to the *markAsReceived*.

After receiving the patient's response, the ExtendConsentExpiration event (Listing 16) will be fired if the variable *consentRenewalState(pc1)* is Approved and *isConsentExpired(pc1)* is TRUE. The *isConsentExpired(pc1)* as a boolean will then be changed from TRUE to FALSE according to the action. Otherwise, the DeletePatientData event (Listing 17) will be fired to add the *pc1* to *markAsDeleted*. In both cases, the consent renewal request is determined by the legal staff.

6. Model evaluation in Event-B

The refined models are formalized and proved correct using the Rodin Platform. The Rodin Platform generates the POs that can be proved automatic or manual. Moreover, it guarantees that all events preserve invariants whenever state variables have changed. The proving results (Table 2) demonstrate that all models were proved automatically by Atelier B provers. Moreover, there were no invariant violations or deadlocks found. The proposed models are publicly accessible at https://github.com/cucpbioinfo/ConsentBasedPrivacy.

ExtendConsentExpiration $\widehat{=}$

Any s, $pc1$ **Where**

$grd1 : s \in dom(sessions) \wedge sessions(s) \in dom(userRoles)$

$grd2 : \exists r \cdot r \in userRoles[sessions[\{s\}]] \wedge r = LegalStaff$

$grd3 : pc1 \in dom(consentRenewalState) \wedge consentRenewalState(pc1) = Approved$

$grd4 : pc1 \subseteq markAsReceived \wedge pc1 \in dom(isConsentExpired) \wedge$

$isConsentExpired(pc1) = TRUE$

$grd5 : isConsentExpired \Leftarrow \{pc1 \mapsto FALSE\} \in$

$(PATIENTS \leftrightarrow CONSENTS) \rightarrowtail\!\!\!\!\!\!\!+ \; BOOL$

Then

$act1 : isConsentExpired(pc1) := FALSE$

End

Listing 16. The ExtendConsentExpiration event.

DeletePatientData $\widehat{=}$

Any s, $pc1$ **Where**

$grd1 : s \in dom(sessions) \wedge sessions(s) \in dom(userRoles)$

$grd2 : \exists r \cdot r \in userRoles[sessions[\{s\}]] \wedge r = LegalStaff$

$grd3 : pc1 \in dom(consentRenewalState) \wedge consentRenewalState(pc1) = Rejected$

$grd4 : pc1 \subseteq markAsReceived \wedge pc1 \in dom(isConsentExpired) \wedge$

$isConsentExpired(pc1) = TRUE$

$grd5 : markAsDeleted \cap pc1 = \varnothing$

Then

$act1 : markAsDeleted := markAsDeleted \cup pc1$

End

Listing 17. The DeletePatientData event.

Table 2
The summary of proof statistics by the Rodin platform for the proposed four consent management state machines based on Event-B models.

Machine name	Number of proof obligations	Automatic (%)	Manual (%)
RPSM	42	42(100%)	0(0%)
WASM	16	16(100%)	0(0%)
PASM	16	16(100%)	0(0%)
CRSM	22	22(100%)	0(0%)

7. Event-B model transformation to class diagram

The proposed models implemented by Event-B can be used as a guideline for software development on the aspects of consent management. According to the object-oriented approach, a class diagram is a static structural model which describes the system's classes, attributes, operations, and associations. It helps developers understand a system's overall structure. Here, we give an example of how to transform our Event-B models into a class diagram. First, identify the primary classes of the system which appear in static variables of Event-B (e.g., sets, constants, variables, and definitions). Second, identify the relation between self or other sets which appear in invariants, which indicate the association between classes. Third, identify events as operations in classes. Also, notice that a transition can be fired, and only if guard conditions are true, then an event occurs. Each guard condition must be implied as a precondition of a method in a class. The globally declared static variables can be mapped to concrete classes (e.g. *AuthorizedUser*, *Role*, *Consent*, *DataSubject*, *DataSubjectConsent*), as can be seen in Fig. 6. The set of PATIENTS represents data subjects under GDPR. We could define a *DataSubject* associated with

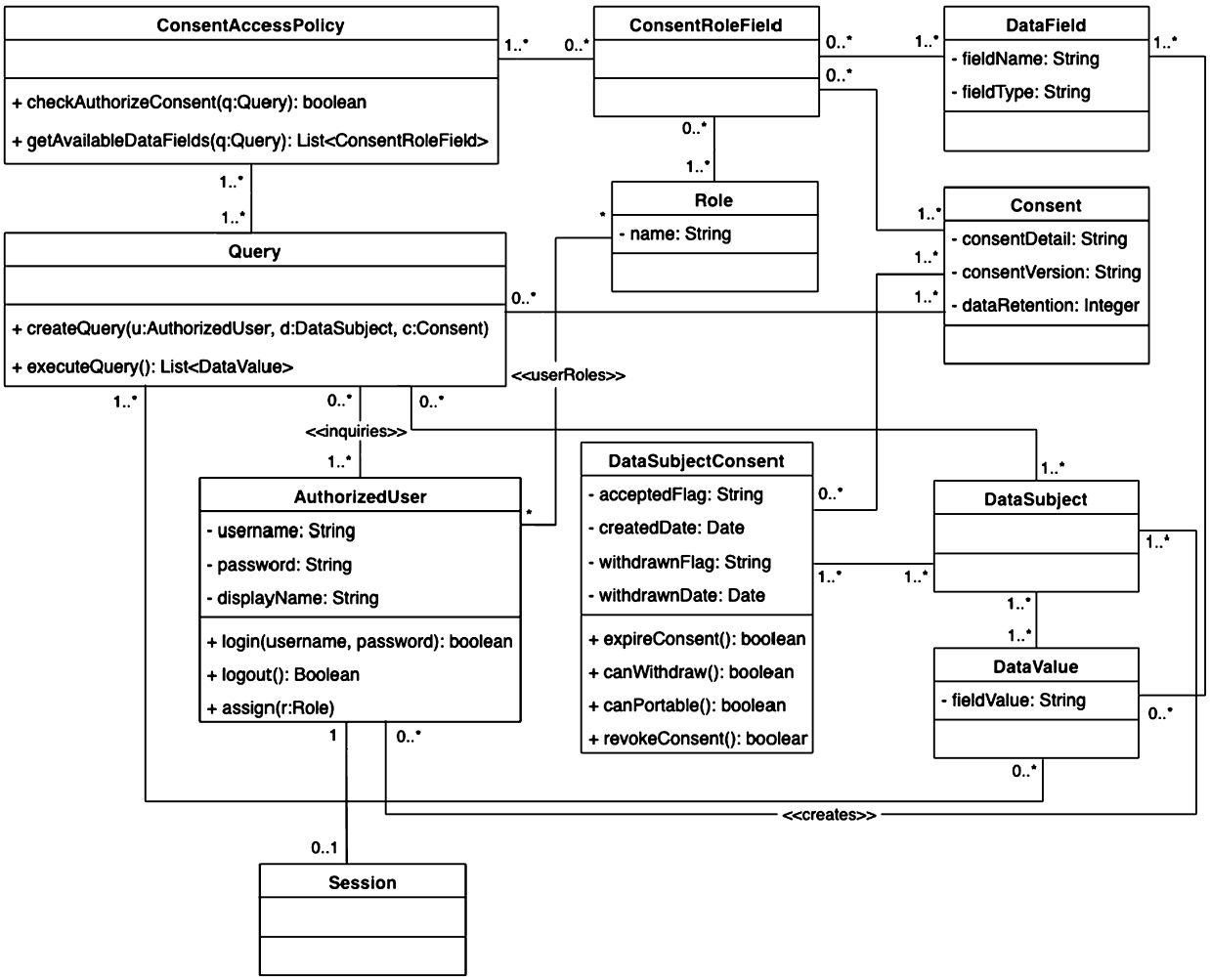


Fig. 6. A class diagram transformed from the proposed consent-based models in Event-B.

DataField, and *DataValue* classes to hold patient personal data. Moreover, GDPR requires the systems to get consent from data subjects before processing data. So, the *Consent* class needs to be created with a set of properties (e.g., consentDetail, dataRetention (in months), consentVersion, createdDate). In the CheckAuthorizeConsent event of RSPM, the variable *consentExpired* is a flag indicating if the data's age exceeds the applicable data retention, defined inside the *ConsentAccessPolicy* class. We need to create a *DataSubjectConsent* class to hold the properties required for calculating the *consentExpired* flag. For example, suppose that we define properties as follows: 1) the *acceptedFlag* indicates a data subject's response to the consent extension, which can be either approved ("Y") or rejected ("N"), 2) the *createdDate* represents the data subject's last response date, 3) the *dataSubject* object indicates this data subject, and 4) the *consent* object indicates the consent that has been approved or rejected by the data subject. To calculate the *consentExpired* flag, we need to retrieve the *DataSubjectConsent* object associated with a specific data subject and consent. After getting the object, check if the *acceptedFlag* = "Y" and *getSystemDate()* > *addMonths(createdDate, consentObject.dataRetention)*, then set the *expiredFlag* = "Y", otherwise set the *expiredFlag* = "N". Our proposed models based on Event-B method were designed to be simple and applicable, which could be easily mapped to the real codes. In the case of RUN-ONCO, a web-based application, we adopted the functionality from the *ConsentAccessPolicy* class (Fig. 6) and enhanced it into *AuthenticationService* and *PatientService* classes (Fig. 1) to make clean and reusable codes.

In addition, particular businesses or systems can also use these models. According to the class diagram in Fig. 6, the *DataSubject* class represents an individual that can recognize a person's uniqueness (e.g., customers, patients, employees). Hence a system has to define a set of data fields of personal data on which can dynamically be added into the *DataField* class (e.g., full name, social security number, birthdate). Since data fields have been defined, a stakeholder who is involved in a software system (e.g., an individual, team, organization) needs to add consent into the *Consent* class and establish a relationship between these data fields. To limit data access precisely, a stakeholder needs to be assigned suitable user roles based on consent data. When collecting personal data, a system needs to obtain the value of personal data in the *DataValue*

class followed by predefined data fields according to a given consent. This research showed that our formal models support the commonly used features of consent management.

8. Conclusion

PbD aims to design and implement a system that supports privacy principles. Several publications considered data privacy as a part of the design and implementation. However, it is difficult for developers to understand and build a system based on abstract concepts in the real world. To make these concepts more concrete, we proposed a set of privacy-enabled formal models, RPSM, WASM, PASM, and CRSM, focusing on aspects of consent management in software development. The correctness of the proposed models was proved by the Event-B method, which combines mathematical techniques based on predicate logic and set theory. With the easy mapping to the actual codes, the models provide the developers with the formally verified guidelines of how to implement a system that supports PbD and GDPR from the aspects of consent management. While the proposed models were demonstrated via the healthcare use case, they can be applied to any business model and help minimize data breaches for both the private and public sectors.

We will focus on the automated transformation of Event-B models into actual codes based on proposed models for future work. This automation will allow developers to automatically build a source code to preserve safety properties for consent management functionality. Moreover, we will explore data sharing based on blockchain among different services. The use of blockchain in data sharing aims to record the request-response interaction between requesters and responders. Moreover, to manage the request-response interaction into a blockchain, we will model smart contracts to extend blockchain features for managing consent and validating consent authorization. By adopting specifications from previous refined models, smart contracts with consent management functionality can restrict access to sensitive or personal data based on the given consent. Finally, to ensure the correctness of smart contracts, we will formalize smart contracts against the state machines.

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.

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] A. Cavoukian, Privacy by Design: The 7 Foundational Principles, August 2009, revised: January 2011.
[3] 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.
[4] 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.
[5] 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.
[6] 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.
[7] T. Stouppa, Phinikiand Studer, A formal model of data privacy, in: I. Virbitskaite, A. Voronkov (Eds.), Perspectives of Systems Informatics, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 400–408.
[8] 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.
[9] S. Matwin, A. Felty, I. Hernádvölgyi, V. Capretta, Privacy in data mining using formal methods, in: Proceedings of the 7th International Conference on Typed Lambda Calculi and Applications, TLCA'05, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 278–292.
[10] C.D. Team, The Coq Proof Assistant Reference Manual: Version 8.13.1, 2021, https://github.com/coq/coq/releases/download/V8.13.1/coq-8.13.1-reference-manual.pdf.
[11] M. Rhoen, Beyond consent: improving data protection through consumer protection law, Int. Policy Rev. 5 (1) (2016) 1–15, https://doi.org/10.14763/2016.1.404.
[12] S. Tokas, O. Owe, A formal framework for consent management, in: A. Gotsman, A. Sokolova (Eds.), Formal Techniques for Distributed Objects, Components, and Systems, Springer International Publishing, Cham, 2020, pp. 169–186.
[13] S.I. Besik, J.-C. Freytag, A formal approach to build privacy-awareness into clinical workflows, SICS Softw.-Intensive Cyber-Phys. Syst. 35 (2019) 141–152, https://doi.org/10.1007/s00450-019-00418-5.
[14] A. Abe, A.C. Simpson, Formal models for privacy, in: EDBT/ICDT Workshops, 2016.
[15] D.F. Ferraiolo, D.R. Kuhn, Role-based access controls, CoRR, arXiv:0903.2171, 2009.
[16] Q. Ni, E. Bertino, J. Lobo, C. Brodie, C.-M. Karat, J. Karat, A. Trombeta, Privacy-aware role-based access control, ACM Trans. Inf. Syst. Secur. 13 (3) (Jul. 2010), https://doi.org/10.1145/1805974.1805980.
[17] P. Ashley, S. Hada, G. Karjoth, C. Powers, M. Schunter, Enterprise Privacy Authorization Language (EPAL), 01 2003.
[18] F. Bu, N. Wang, B. Jiang, H. Liang, "Privacy by Design" implementation: information system engineers' perspective, Int. J. Inf. Manag. 53 (2020) 102124, https://doi.org/10.1016/j.ijinfomgt.2020.102124.
[19] Y. Tian, Y. Li, X. Liu, R.H. Deng, B. Sengupta, PriBioAuth: privacy-preserving biometric-based remote user authentication, in: 2018 IEEE Conference on Dependable and Secure Computing (DSC), 2018, pp. 1–8.
[20] J.-R. Abrial, Modeling in Event-B: System and Software Engineering, 1st edition, Cambridge University Press, USA, 2010.
[21] 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.

[22] E. Vanezi, D. Kouzapas, G.M. Kapitsaki, T. Costi, A. Yeratziotis, C. Mettouris, A. Philippou, G.A. Papadopoulos, GDPR compliance in the design of the INFORM e-learning platform: a case study, in: 2019 13th International Conference on Research Challenges in Information Science (RCIS), 2019, pp. 1–12.
[23] E. Politou, E. Alepis, C. Patsakis, Forgetting personal data and revoking consent under the GDPR: challenges and proposed solutions, J. Cybersecurity 4 (1) (03 2018), https://doi.org/10.1093/cybsec/tyy001, https://academic.oup.com/cybersecurity/article-pdf/4/1/tyy001/27126900/tyy001.pdf.
[24] J.-R. Abrial, S. Hallerstede, Refinement, decomposition, and instantiation of discrete models: application to Event-B, Fundam. Inform. 77 (1–2) (2007) 1–28.
[25] 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).
[26] 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.
[27] J. Rest, D. Boonstra, M. Everts, M. van Rijn, R. Paassen, Designing Privacy-by-Design, 2014, pp. 55–72.
[28] 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.