A Formal Modeling Scheme for Analyzing a Software System Design against the GDPR

Evangelia Vanezi, Georgia M. Kapitsaki, Dimitrios Kouzapas and Anna Philippou
Department of Computer Science, University of Cyprus, Nicosia, Cyprus

Keywords: GDPR, Privacy Protection, π-calculus, Static Analysis, Privacy by Design.

Abstract: Since the adoption of the EU General Data Protection Regulation (GDPR) in May 2018, designing software systems that conform to the GDPR principles has become vital. Modeling languages can be a facilitator for this process, following the principles of model-driven development. In this paper, we present our work on the usage of a π-calculus-based language for modeling and reasoning about the GDPR provisions of 1) lawfulness of processing by providing consent, 2) consent withdrawal, and 3) right to erasure. A static analysis method based on type checking is proposed to validate that a model conforms to associated privacy requirements. This is the first step towards a rigorous Privacy-By-Design methodology for analyzing and validating a software system model against the GDPR. A use case is presented to discuss and illustrate the framework.

1 INTRODUCTION

Privacy protection is now more crucial than ever, and this is reflected in the new EU General Data Protection Regulation (GDPR) (European Parliament and Council of the European Union, 2015). The GDPR has been applied since May 25th, 2018 not only to organizations established in the EU, but also to non-EU established organizations that process personal data[1] of individuals who are located in the EU. The GDPR defines multiple principles and rights for EU residents in regard to personal data collection, storage and processing. All existing systems must be reviewed and adapt their processes accordingly, and new systems must be built and function in a way that ensures their GDPR compliance.

As this change is recent, there have only been few attempts to understand the GDPR, mainly from the legal perspective, and even fewer discussing mechanisms to support GDPR compliance (Gjermundrød et al., 2016; Hintze and LaFever, 2017). In the context of this work, our aim is to discuss the GDPR with regard to data collection and processing as this need arises in software systems, and to propose a rigorous methodology for embedding GDPR compliance into a software engineering development methodology, specifically into the design and modeling phase, using a π-calculus (Milner et al., 1992) based formal modeling scheme exploiting the Privacy calculus (Kouzapas and Philippou, 2017). Additionally, we present an extension to the Privacy calculus integrating the notion of granting and withdrawing consent. The ulterior, long-run objective of our work is to provide a rigorous framework for developing GDPR-compliant systems that can support the entire software development methodology, thus pursuing conformance to Privacy by Design (PbD) or, as referred to in the GDPR, Data Protection By Design (Rubinstein, 2011).

Specifically, in this paper we describe the integration of the following principles of the GDPR within a formal framework for reasoning about privacy-related properties: *lawfulness of processing* by providing consent, *consent withdrawal*, and *right to erasure*. In particular, we build on the Privacy calculus, a process calculus based on the π-calculus with groups extended with constructs for reasoning about private data (Cardelli et al., 2000). We extend the Privacy Calculus syntax and semantics, in order to include the aforementioned GDPR principles. Furthermore, we associate the calculus with a type system that validates the proper processing of data inside a system and we prove that a well-typed system satisfies the above mentioned GDPR requirements. We argue that the resulting framework can be used for providing models of software systems while supporting the static analysis of GDPR compliance during the design

[1]As defined in GDPR Article 4(1), personal data is any information relating to an identified or identifiable natural person ('data subject')

Vanezi, E., Kapitsaki, G., Kouzapas, D. and Philippou, A.
A Formal Modeling Scheme for Analyzing a Software System Design against the GDPR.
DOI: 10.5220/0007722900680079
In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2019)*, pages 68-79
ISBN: 978-989-758-375-9

phase. Furthermore, we discuss the possibility for future additions to the framework, incorporating more GDPR provisions, in order to support an almost fully compliant software model design and verification.

We consider the type system accompanying our framework to be a promising dimension of our approach, since it enhances the methodology with the ability to statically validate models against formally-defined privacy policies. This approach is complementary to verification by model-checking and has the advantage that it does not require the construction of a system's state space, which may yield the state-space explosion problem. Moreover, this dimension of our proposal can be transferred to the development phase of software systems through proper tools towards the generation of GDPR-compliant code. Thus, as future work we envision the development of the framework into programming semantics and analysis tools for supporting the software engineering development phase and the construction of privacy-respecting code following the Privacy by Design principle.

The remainder of the paper is structured as follows. In Section 2, we discuss some background work: we present the Principle of Data Protection by Design and the use of modeling languages during the design phase of the software development cycle. We then provide a quick overview of the π-calculus and the Privacy calculus. In Section 3, we introduce the Privacy calculus as a language to model a software system handling private data, and in Section 4 we propose an extension to the Privacy calculus that incorporates the GDPR principles of lawfulness of processing by providing consent, consent withdrawal and right to erasure into the software's model. Section 5 presents this framework formally and it develops a type system for guaranteeing compliance to privacy-related properties pertaining to the GDPR principles under investigation. Finally, Section 6 concludes the paper and discusses possible directions for future work. Due to lack of space, proofs of results are omitted. Furthermore, for the complete exposition to the Privacy calculus the reader is referred to (Kouzapas and Philippou, 2017).

2 BACKGROUND AND RELATED WORK

2.1 Data Protection by Design

Privacy by Design was introduced by the Dutch Data Protection Authority (Data Protection and Privacy Commissioners, 2010) and the former Information and Privacy Commissioner of Ontario, Canada, Ann Cavoukian (Cavoukian, 2008) having as a result of their joint project a published report (Hes and Borking, 1998) as explained in (Hustinx, 2010). It advocates that privacy should be incorporated into systems by default and should be a priority from the beginning of a system's design. Similarly, the GDPR (European Parliament and Council of the European Union, 2015) defines *Data Protection by Design* in Article 25(1) as *"implementing the appropriate technical and organizational measures in order to meet the requirements of the Regulation and protect the rights of data subjects"*. Consequently, the obligation of software developers to build systems designed to be compliant to the GDPR becomes apparent.

By its definition, Data Protection by Design needs no specific distinct analysis, handling and mechanisms in order to be fulfilled. If all principles and rights are correctly embedded in a system's specifications model and are transferred into the succeeding implementation, then Data Protection by Design will also be guaranteed. Moreover, Data Protection by Design advocates the proactive consideration of privacy issues: ensuring a software' compliance should take place from the design phase and possible pitfalls should be anticipated and prevented before they can materialise, rather than remedied on a reactive basis.

In the literature, different approaches to PbD have been proposed. *PriS* is a security requirements engineering method that integrates privacy requirements in the early stages of the system development process (Kalloniatis et al., 2008). It focuses on the impact of privacy requirements onto the organizational processes and on suggesting implementation techniques for realizing these requirements. *privacyTracker* considers the GDPR and proposes a framework that supports some of its basic principles including data traceability and allowing a user to get a cryptographically verifiable snapshot of her data trail (Gjermundrød et al., 2016). PbD for Internet of Things and relevant steps that should be followed for assessing applications is discussed in (Perera et al., 2016).

2.2 Software Design and Modeling

During the design phase, a software system model is usually developed capturing the requirements specified during the requirements analysis step. Such a model presents a view of the software at a suitable level of abstraction by visualizing, specifying and documenting all artifacts and aspects of software systems. To this effect, modeling languages have been developed and are employed for providing system models, a notable example being the Unified Modeling Language (UML) (Fowler, 2004). Extensions to UML have been proposed, such as UMLsec that allows to express security relevant information for a system specification (Jürjens, 2002), the Privacy-aware Context Profile (PCP) that can be exploited in context-aware applications (Kapitsaki and Venieris, 2008) and the Privacy UML profile that can be used to capture the privacy policies of a software system (Basso et al., 2015). These models can be subsequently used to test whether the design meets the system specifications, but also to produce the system source code skeleton following the paradigm of Model Driven Engineering (MDE) (Schmidt, 2006; Kapitsaki et al., 2008). A previous work using modeling to analyze the system design and propose security and privacy controls that can improve it can be found in (Ahmadian et al., 2018). In this framework, a privacy impact assessment (PIA) methodology is introduced. Other works rely on ontologies and modeling in the Web Ontology Language (OWL) or the Resource Description Framework (RDF), such as Linked USDL Privacy (Kapitsaki et al., 2018). However, due to their semi-formal nature and lack of precise semantics, modeling languages as the above do not support the formal verification of property satisfaction of system models.

Formal methods aim to address this challenge by providing languages and tools for the rigorous specification, development and verification of systems. In this paper, we focus on one such formalism, namely the π-calculus and we argue that it can provide the basis of a modeling language for software systems that may additionally support verification, including the validation of conformance to privacy requirements from the design phase of a software system therefore fulfilling the Data Protection by Design GDPR requirement. The π-calculus has been used for formalizing aspects of the UML modeling language (Lam, 2008), as well as for transforming the basic workflow patterns of Business Process Model and Notation (BPMN) into equivalent π-calculus code (Bousetoua et al., 2015). It has also formed the theoretical basis of the Business Process Modeling Language (BPML) (Havey, 2005) and of Microsoft's XLANG (Thatte, 2001).

2.3 Modeling Software in a π-calculus based Language

The π-calculus is a simple, concise but very expressive process calculus that can describe concurrent processes whose configuration may evolve during computation, such as mobile and web applications (Milner et al., 1992). It is associated with formal semantics which precisely define the behavior of a system, based on which dynamic (e.g. model checking) and static (e.g. type checking) analyses of systems properties can be performed (Sangiorgi and Walker, 2003). A software system model expressed in a π-calculus based language will include representations of all components and entities of the system as processes that are able to interact and communicate with each other through communication channels. Channels can by dynamically created and even sent from one process to another creating new communication links between the processes, thus enabling the dynamic evolution of a system.

Table 1: Privacy Calculus Basic Syntax.

Functionality	π-Calculus Term	
Concurrency	$P \mid Q$	(1)
Input	$c?(x).P$	(2)
Output	$c!\langle v \rangle.P$	(3)
Restriction	$(\nu\, n)P$	(4)
Replication	$*P$	(5)
Matching	`if` e `then` P `else` Q	(6)
Nil (stopped) process	$\mathbf{0}$	(7)
Store	$\bar{r} \triangleright [\iota \otimes \delta]$	(8)
Systems	$\mathsf{G}[P] \mid \mathsf{G}[S] \mid S \parallel S \mid (\nu\, n)S$	(9)

The core functionalities provided by the basic π-calculus syntax are summarized in Table 1, lines (1)-(7). Concurrent composition $P \mid Q$ allows the parallel execution of processes, i.e. system entities. Similarly to software systems, concurrent processes may proceed independently, each executing its own code, while communicating with each other on shared channels referred to as *names*. In its basic form, the π-calculus supports synchronous communication which occurs through input and output of messages on channels. The input term, $c?(x).P$, describes a process waiting to receive a message on channel c, resulting in variable substitution storing the message in x, before proceeding as P. The output construct, $c!\langle v \rangle.P$, describes a process sending v as a message on channel c and then proceeds as P. Restriction, or new-name creation, $(\nu\ n)P$, allows for the allocation of a new name n inside the scope of process P. Replication, $*P$, allows the creation of multiple copies of process P. Matching or conditional construct, `if` e `then` P `else` Q, can evolve as P if the condition e is evaluated as true, or as Q otherwise. Finally, the terminated process, **0**, defines a process that has no interactions.

The operational semantics of the π-calculus consists of a set of rules that describe the behavior of each of the above constructs, thus allowing to construct the possible executions of a defined system. For instance, in the case of the parallel composition construct, it specifies that whenever two processes are able to send/receive a message respectively on the same channel, then computation proceeds by the exchange of the message in a synchronized step after which the processes will evolve to their subsequent state. As an example, consider the following communication implementing the transmission of value 42 on channel a between two concurrent processes, where $\longrightarrow$ captures the evolution and where τ specifies the name of the transition being an internal communication.

$$a!\langle 42\rangle.P \,|\, a?(x).Q \xrightarrow{\tau} P \,|\, Q\{^{42}/_{x}\}$$

Note that, on the receiving side after the interaction the value 42 is substituted in the process continuation on variable x as described by the notation $Q\{^{42}/_{x}\}$.

2.4 The Privacy Calculus

Many different extensions of the π-calculus have been considered in the literature. Among these works, numerous studies have focused on security and privacy properties. One such extension created with the purpose of describing and analyzing cryptographic protocols is the spi-calculus (Abadi and Gordon, 1997), whereas in (Cardelli et al., 2000) the π-calculus is extended with the notion of groups as types for channels which are used to statically prohibit the leakage of secrets. Type systems have also been employed in process calculi to reason about access control that is closely related to privacy. For instance, the work on the Dπ-calculus has introduced sophisticated type systems for controlling the access to resources advertised at different locations (Hennessy et al., 2005; Hennessy and Riely, 2002; Hennessy, 2007). Furthermore, discretionary access control has been considered in (Bugliesi et al., 2009) which similarly to our work employs the π-calculus with groups, while role-based access control (RBAC) has been considered in (Braghin et al., 2006; Dezani-Ciancaglini et al., 2010; Compagnoni et al., 2008). In addition, authorization policies and their analysis via type checking has been considered in a number of papers including (Fournet et al., 2007; Backes et al., 2008; Bengtson et al., 2011).

In this paper, we extend the work of (Kouzapas and Philippou, 2017), where the authors propose a formal framework, the Privacy calculus, for capturing privacy requirements. This framework is based on (Cardelli et al., 2000). Specifically, as shown in Table 1, lines 8 and 9, the Privacy calculus introduces the notion of personal data, $\mathsf{id} \otimes c$ that associate identities, id, with values, c. It also extends process terms with stores of private data $\overline{r} \triangleright [\mathsf{id} \otimes c]$, which are processes that store and provide access to private data. These stores are accessed through names, r, called references. Processes are organized in groups, $\mathsf{G}[P]$, which associate each entity P with an identifier/role G in the system.

As such, personal data as defined in the GDPR can be represented in the Privacy calculus by associating each piece of plain data with an identifier value, $\mathsf{id} \otimes c$. The privacy calculus also enables the form $\_ \otimes c$ to associate data with a hidden identity. The collection of stores in a system can be thought of as a database table, managed by a database manager process. Other processes can hold a reference to a store, to access, read, and process the data. The type system ensures that data manipulation conforms to a system policy, which specifies how each process/role in the system may process private date.

Furthermore, the Privacy calculus provisions allow the analysis of privacy-related requirements. To begin with, the notion of a group ensures secrecy. This is achieved by associating channels with groups, which are types for channels. No channel of a certain group can be communicated to processes outside the scope of the group, as in (Cardelli et al., 2000). This task can be checked statically. The Privacy calculus goes a step further by associating processes with groups, and uses the group memberships of processes to distinguish their roles within systems. The framework is accompanied by a type system for capturing privacy-related notions and a privacy language for expressing privacy policies. The policies enable to specify how the various agents, who are referred to by their groups, or roles, may or may not handle the system's sensitive data. In the current work we will not be defining privacy policies in contrast to (Kouzapas and Philippou, 2017). Instead, the proposed validation by type checking will specifically analyze conformance to the specific GDPR principles and rights under discussion.

3 A SOFTWARE SYSTEM SPECIFICATION MODEL CASE

In order to present the rationale of our approach, assume that we want to build a part of an online banking system with the following specifications:

1. The system waits to receive the following data from the user: her phone number.

2. The user's phone number is classified as personal data.

3. The system receives and saves the data provided by the user to the system database.
4. When a transaction occurs, a notification system process reads the user's phone number from the database and sends a text message to the number. The system's functionality is then terminated.

These specifications can be modeled in the Privacy calculus as follows. A user component will provide a private data input. The system entities will receive the input, manage the input's storage, read the data stored, and manipulate them. Initial communication between the user and the system is realized through channel c. Groups will be used in order to define the distinct system entities, i.e. the interacting subsystems expressing different functionality. A user entity process, U, of group User, creates the private session communication channel a, and shares it with the system. Next, she waits to receive on channel a a store reference and, when received, she writes her phone number on it.

The functionalities of writing and reading data directly to or from the store are considered black boxes. This is a typical practice during the modeling phase, when some aspects of a system are abstracted and only become concrete during the implementation phase. As such, the abstractions of writing and reading directly from the store will be implemented during the system development, for instance, by calling a respective RESTful service that offers an interface for communicating with the system data store. In addition, these abstractions enable the *Right of Access* of the data subject to her private data as required by the GDPR. The behavior of the user can thus be modeled as follows:

$$U \ ::= \ (\nu\, a)(\mathsf{User}[c!\langle a\rangle.a?(x).x!\langle \mathsf{id}\otimes\mathtt{phone}\rangle.P'])$$

The system entity DBase of group DBaseService is responsible for holding and managing the data storage, i.e. the user's store. For the purpose of the example we will assume that a store already exists for the specific user, filled with the user identifier and a placeholder for the phone number:

$$\mathsf{DBase} \ ::= \ \mathsf{DBaseService}[\overline{r}\triangleright[\mathsf{id}\otimes *]]$$

Another system entity S_1 of group InterfaceService, already considered to be holding a reference to the store, waits for an input from the user on the public communication channel c to be used for establishing a private communication session between the two entities. It then proceeds to send the store reference to the user through that channel:

$$S_1 \ ::= \ \mathsf{InterfaceService}[c?(x).x!\langle r\rangle.\mathbf{0}]$$

Finally, the system entity S_2 of group NotificationService, also considered to be holding the reference to the user's store, takes as input from the store the respective data, and uses it to send a text notification to that phone number. This process will be triggered when a new transaction occurs and can be modeled as follows:

$$\begin{aligned} S_2 \quad ::= \quad & \mathsf{NotificationService}[r?(x\otimes y).\\ & y!\langle\mathtt{notification}\rangle.\mathbf{0}] \end{aligned}$$

The whole system is defined as the parallel composition of the above processes comprising a system of group

$$\mathit{System} \ ::= \ \mathsf{Sys}[U\,|\,\mathsf{DBase}\,|\,S_1\,|\,S_2]$$

To further understand how the Privacy calculus models interact, we proceed to provide an execution of the system.

$$\begin{aligned} \mathit{System} \xrightarrow{\tau}\ & \mathsf{Sys}[(\nu\, a)(\mathsf{User}[a?(x).x!\langle\mathsf{id}\otimes\mathtt{phone}\rangle.P'] \\ & |\,\mathsf{InterfaceService}[a!\langle r\rangle.\mathbf{0}])\,|\,\mathsf{DBase}\,|\,S_2] \\ \xrightarrow{\tau}\ & \mathsf{Sys}[(\nu\, a)(\mathsf{User}[r!\langle\mathsf{id}\otimes\mathtt{phone}\rangle.P'] \\ & |\,\mathsf{InterfaceService}[\mathbf{0}])\,|\,\mathsf{DBase}\,|\,S_2] \\ \xrightarrow{\tau}\ & \mathsf{Sys}[(\nu\, a)(\mathsf{User}[P']\,|\,\mathsf{InterfaceService}[\mathbf{0}]) \\ & |\,\mathsf{DBaseService}[\overline{r}\triangleright[\mathsf{id}\otimes\mathtt{phone}]]\,|\,S_2] \\ \xrightarrow{\tau}\ & \mathsf{Sys}[(\nu\, a)(\mathsf{User}[P']\,|\,\mathsf{InterfaceService}[\mathbf{0}]) \\ & |\,\mathsf{DBaseService}[\overline{r}\triangleright[\mathsf{id}\otimes\mathtt{phone}]] \\ & |\,\mathsf{NotificationService}[\mathtt{phone}!\langle\mathtt{notification}\rangle.\mathbf{0}]] \end{aligned}$$

The first step captures the synchronization between the user process and S_1, where the former send the fresh channel a through public channel c resulting in a substitution within S_1 of variable y by name a. In the second step, S_1 sends the store reference r to the user process, with reference r substituting variable x and subsequently used by the user to update her private information on the store managed by the DBaseService. After the interaction we can observe that the empty store now contains the *phone* of the user. Finally, process S_2 receives the phone number of the user from the database and is ready to send a notification to the user via her phone number.

4 GDPR AND THE PRIVACY CALCULUS

The GDPR defines multiple principles and rights regarding personal data collection, storage, and processing. While these principles were mostly welcomed, various challenges have been recognized in terms of effectively integrating the new requirements into computing infrastructures. Furthermore, a question that arises is how to verify that a system adheres to privacy requirements as enunciated by the GDPR. In this work, we address this challenge by providing an initial step towards a formal framework for reasoning about privacy-related concepts. On the one hand, such a framework would provide solid foundations towards understanding the notion of privacy while, on the other hand, it would provide the basis for system modeling and analysis tools for privacy-respecting code. Beyond proposing the incorporation of such a framework into a software's engineering methodology, we concentrate on GDPR Article 6, *Lawfulness of Processing*, and the requirements defined therein regarding *consent*, Article 7(3), *Consent Withdrawal* and Article 17, *The Right to Erasure*. We then evaluate the Privacy calculus as a formalism to reason about the GDPR, we discuss necessary additions to address the above articles and we demonstrate them with the use of examples. In the next section we formally define these extensions and present a static methodology for assessing the conformance of a system to these requirements.

4.1 Storing and Processing Personal Data

We recall that according to the first principle of Article 5 (*"Lawfulness, fairness and transparency"*), Article 6 (*"Lawfulness of Processing (1)"*), and in Recital 40 (*"Lawfulness of data processing"*), private data may be collected, stored, and processed by a system, only if the users (data owners) have given their consent for the intended collection and processing. The consent should be given explicitly, freely and willingly by the user, based on one or more specific purposes stated and described in the privacy policy.

In order to reason about the above concepts, a formalism should contain the notion of *private data* as a basic entity. This is already the case in the Privacy calculus, where private data are considered to be data structures representing pieces of information related to individuals along with the identities of the associated individuals. Furthermore, the formalism should provide a clear definition of the notion of *consent* and ensure that consent is given before any storage or processing of the data is performed. Naturally, this consent should be given by the owner of the data to be processed.

In order to enable the above, we propose the following extension in Privacy calculus. Initially, the calculus should enable the provision (and withdrawal) of consent, whereas the notion of a user's identifier should become a first-class entity to allow the verification that a consent (or withdrawal of consent) for a piece of private data is indeed given by the appropriate party. To achieve this, we extend the Privacy calculus with (i) additional constructs for providing consent, receiving consent and creating an associated store upon the receipt, and withdrawing consent and emptying the associated store, and (ii) the introduction of special identifiers and their association with system processes. The implementation of the consent construct should ensure that provision of consent should precede any storage of personal data in a store, except when the system is handling anonymized data. No processing of private data will be enabled before the consent of the user. As such, we associate the grant of consent by a user with the creation of a store within the software. Once consent is obtained, the data is saved within the store and its reference can be communicated to other entities in the system, which may in turn process the data as needed by the software specification and specified by its privacy policy. Furthermore, the store reference is shared with the consenting entity, i.e. the user, giving the user direct access to the data associated with her within the system at any point in time.

To ensure the association between a process entity and its associated data, a user process is annotated by a unique identifier value, $\{P\}_{\mathsf{id}}$, where id is embedded within all communications requiring the matching between a data owner and their data. Note that while the actions of providing consent and writing personal data to the store are implemented by single constructs within the calculus, they should be thought of as black boxes including functionality that will be implemented in detail at coding level. We point out that this abstraction will need to be appropriately implemented for safeguarding identifiers and channel communication from potential hostile entities, possibly by embedding a communication protection mechanism, such as a public-private key cryptography scheme, as in (Abadi and Gordon, 1997). We illustrate the above concepts by extending the example of Section 3.

Example 4.1. *Section 3 model case with the following additions:*

1. *The user should provide consent for the storage and usage of her phone number by the system prior to any storage and processing of the phone number by the system.*
2. *No store will be associated with the user within the system before the user explicitly provides her consent.*

To model these additions, we extend the description of the various entities as follows.

User entity process U is annotated by its identifier. Furthermore, once a private session is achieved with the database manager it provides its consent via channel a while simultaneously receiving a reference to its newly-created store. This is achieved via construct $a \triangleleft \mathsf{consent}(x)$. As in the previous version of the example, the user proceeds to write her data to the store. This is modeled as follows:

$$\{U\}_{\mathsf{id}} \quad ::= \quad (\nu\, a)(\mathsf{User}[c!\langle a\rangle.a \triangleleft \mathsf{consent}(x).\; x!\langle \mathsf{id} \otimes \mathtt{phone}\rangle.\mathbf{0}])$$

Moving on to the process of receiving and storing the private data of the user by the software system, in the proposed extension a new entity S_1 of group DBInterfaceService is defined, combining the database and the interface service. This process dynamically creates a new reference for a store and provides it to the disposal of the user as soon as she gives her consent (the actual store created is not present in the definition below since it will be dynamically created once consent is given, as implemented by the semantics of our calculus presented in the next section). The fresh store reference needs also to be communicated to the notification service, which is the purpose of the communication on channel d below. This is modeled as follows:

$$S_1 \quad ::= \quad \mathsf{DBInterfaceService}[c?(y).\; (\nu\, r)(y \triangleright \mathsf{consent}(r).d!\langle r\rangle.\mathbf{0})]$$

System process S_2 receives the reference to the store and then continues to perform its task as in the previous example:

$$S_2 \quad ::= \quad \mathsf{NotificationService}[d?(p).p?(x \otimes y).\; y!\langle \mathtt{notification}\rangle.\mathbf{0}]$$

The system definition as a whole is comprised of the parallel composition of the three processes:

$$\mathit{System} \;::=\; \mathsf{Sys}[\{U\}_{\mathsf{id}} \,|\, S_1 \,|\, S_2]$$

4.2 Withdrawing Consent and Forgetting Personal Data

The Right to Erasure is stated in Article 17 of the GDPR, defining the right of a user to have her personal data erased from a system without any undue delay, if one of multiple reasons stated is applied, including consent withdrawal. As defined in GDPR Article 7 (3): "*the data subject shall have the right to withdraw his or her consent at any time*". In this case, the individual's personal data should be deleted or become anonymized in a way that the data subject can no longer become identified by them.

In this work, we examine consent withdrawal as the cause for erasure enforcement, though the machinery can be easily applied to other erasure causes. In this case, we may assume that consent withdrawal originates directly from the user process and, given such a directive, the software is obliged to forget the user's data. Thus in the Privacy calculus, when a deletion instruction is given, the associated store will be emptied with the consequence of having its reference leading to no data. Subsequently, no data will be available to any entity possessing a reference to the store and attempting to read or write to it. The above mentioned functionality is implemented by appropriate rules added to the Privacy calculus semantics, and specifying an abstraction of a GDPR-compliant behavior to be appropriately instantiated in the implementation process.

Example 4.2. *Let us consider Example 4.1 where a user provided consent and sent her phone number to a system. To implement the Right to Erasure, let us assume that the user has the right at a later stage to withdraw her consent. As such, the example must be expanded with the following specifications:*

1. *The user should be able to withdraw her consent.*
2. *When a user's consent is withdrawn all her personal data should be deleted.*
3. *No further storage or processing of the specific user's personal data is allowed unless the user provides again her consent.*

The model of a user process $\{U\}_{\mathsf{id}}$ as in Example 4.1 who withdraws her consent at some point after submitting her data can be modeled as follows:

$$\{U\}_{\mathsf{id}} \quad ::= \quad (\nu\, a)(\mathsf{User}[c!\langle a\rangle.a \triangleleft \mathsf{consent}(x).\; x!\langle \mathsf{id} \otimes \mathtt{phone}\rangle.x \triangleleft \mathsf{withdraw}.\mathbf{0}])$$

The semantics of consent withdrawal instruction ensure that such an instruction will synchronize with the associated store resulting in the automatic deletion of all content within the store causing all attempts to process the store futile (again through appropriate enunciation of the semantics rules for store processing). Note that system entities S_1 and S_2 require no change in their definitions. Furthermore, the entire system's definition remains unaltered.

5 THE EXTENDED PRIVACY CALCULUS

In this section we describe formally the Extended Privacy calculus and its operational semantics and we propose a type-checking methodology for ensuring

Table 2: Extended Syntax of the Privacy Calculus.

Name	Definition
(identity values)	$\iota \; ::= \; \mathsf{id} \mid \_ \mid x$
(data values)	$\delta \; ::= \; c \mid * \mid x$
(private data)	$\iota \otimes \delta$ where $\iota \neq x \Rightarrow \delta = c$ and $\iota = x \Rightarrow \delta = y$
(identifiers)	$u \; ::= \; a \mid r \mid x$
(terms)	$t \; ::= \; a \mid r \mid \iota \otimes \delta \mid d \mid x$
(constants)	$v \; ::= \; a \mid r \mid \mathsf{id} \otimes d \mid d$
(placeholders)	$k \; ::= \; x \mid x \otimes y \mid \_ \otimes x$
(processes)	$P \; ::= \; \mathbf{0} \mid u!\langle t \rangle.P \mid u?(k).P \mid (\nu\, n)P$
	$\mid \; P \,\vert\, P \mid *P \mid \mathtt{if}\ e\ \mathtt{then}\ P\ \mathtt{else}\ P$
	$\mid \; \overline{r} \triangleright [\iota \otimes \delta] \mid u \triangleleft \mathsf{consent}(x).P$
	$\mid \; u \triangleright \mathsf{consent}(r).P \mid u \triangleleft \mathsf{withdraw}.P$
(systems)	$S \; ::= \; \mathsf{G}[P] \mid \mathsf{G}[\{P\}_{\mathsf{id}}] \mid \mathsf{G}[S] \mid S \,\Vert\, S \mid (\nu\, n)S$

that well-typed processes respect the GDPR requirements pertaining to *consent*, as discussed in the previous section. Due to the lack of space we only present the extensions we propose to the Privacy calculus and we refer the reader to (Kouzapas and Philippou, 2017) for its complete exposition.

5.1 Formal Definition

The extended syntax of the Privacy calculus proposed for our modeling scheme can be found in Table 2. The syntax first defines the values used. Assume a set of constants $c \in C$, and a set of variables $x, y \in V$. A special case of values denoted by ι refer to identities. Symbol ι ranges over (i) identities, id, (ii) the hidden identity, $\_$, denoting anonymized data, and (iii) variables x. Data values ranged over by δ include constants, c, the empty constant, $*$, and variables. Private data $\iota \otimes \delta$ associate identities id with data c. Variable placeholders for private data are written as $x \otimes y$.

The set of identifiers used for the definition of the Privacy calculus are defined next. Assume a set of names $n \in N$ that are partitioned over names, a, and store references, r. We use meta-variable u to range over names or variables. Values, ranging over v, include names, private data, or data; placeholders, ranging over k, include the variable structures of the calculus, whereas terms, ranging over t, include both values and placeholders.

The syntax of processes follows. It includes the Privacy calculus constructs together with constructs for creating private data stores upon consent and emptying private data stores upon consent withdrawal. Term $u \triangleleft \mathsf{consent}(x).P$ defines the term, where a process provides with consent on creating a new store on channel u. The dual operation is term $u \triangleright \mathsf{consent}(r).P$, where the process receives a consent request on channel u and creates a new store on reference r. During such an interaction the consenting process ($u \triangleleft \mathsf{consent}(x).P$) will receive the new store reference on variable x. Term, $u \triangleleft \mathsf{withdraw}.P$ denotes a process ready to withdraw consent on a store reference u.

The syntax of systems is also extended with system $\mathsf{G}[\{P\}_{\mathsf{id}}]$ that denotes a system that runs on a process with an identity. Processes with identities are used to abstract individual users within a system, whose private data are being processed.

The computation steps of the Privacy calculus are defined using a relation called *labeled transition relation*. Specifically, we write $S \xrightarrow{\ell} S'$ to denote that a system S can take a step and evolve into system S' by executing the action indicated by label ℓ. Whenever two parallel processes/systems can execute two dual actions then they synchronize with each other and their composition executes a computation interaction, indicated by τ, as follows:

$$\frac{S_1 \xrightarrow{\ell_1} S_1' \quad S_1 \xrightarrow{\ell_2} S_2' \quad \ell_1 \ \mathsf{dual} \ \ell_2}{S_1 \,|\, S_2 \xrightarrow{\tau} S_1' \,|\, S_2'}$$

The labels of the Privacy calculus are extended with the labels:

$$a \triangleleft \mathsf{consent}(r) \quad a \triangleleft \mathsf{consent}(r)@\mathsf{id} \quad a \triangleright \mathsf{consent}(r)@\mathsf{id}$$
$$r \triangleleft \mathsf{withdraw} \quad r \triangleleft \mathsf{withdraw}@\mathsf{id} \quad \overline{r} \triangleright \mathsf{withdraw}@\mathsf{id}$$

where

$$a \triangleleft \mathsf{consent}(r)@\mathsf{id} \ \mathsf{dual} \ a \triangleright \mathsf{consent}(r)@\mathsf{id}$$
$$r \triangleleft \mathsf{withdraw}@\mathsf{id} \ \mathsf{dual} \ \overline{r} \triangleright \mathsf{withdraw}@\mathsf{id}$$

Label $a \triangleleft \mathsf{consent}(r)$ denotes the basic action for providing consent on channel a and receiving a reference on channel r, whereas label $a \triangleleft \mathsf{consent}(r)@\mathsf{id}$ is the same action lifted to a user identity, id. Dually action $a \triangleright \mathsf{consent}(r)@\mathsf{id}$ is the acceptance of a consent on channel a and the creation of a new store with reference r and identity id. Withdraw labels are $r \triangleleft \mathsf{withdraw}$ that denotes a withdraw on reference r; $r \triangleleft \mathsf{withdraw}@\mathsf{id}$ that lifts a withdraw at the user level; and $\overline{r} \triangleright \mathsf{withdraw}@\mathsf{id}$ that denotes the receipt of a withdraw on reference r with identity, id.

The labeled transition semantics definition follows in Table 3. The first rule extends structural congruence with a garbage collection rule to collect stores that do not contain useful private data. Rule [SConsentU1] describes the interaction of a process giving consent on channel a; the process observes the $a \triangleleft \mathsf{consent}(r)$ label. Next, rule [SConsentU2] lifts a consent action to a process associated with an identity, using label $a \triangleleft \mathsf{consent}(r)@\mathsf{id}$. The dual action of receiving consent is done via label $a \triangleright \mathsf{consent}(r)@\mathsf{id}$. After the label is observed a new store, $\overline{r} \triangleright [\mathsf{id} \otimes *]$

Table 3: Additions to Privacy Calculus Labeled Transition Semantics.

Rule Name	Structure
[GarbageC]	$(\nu\ r)(\overline{r} \triangleright [\_ \otimes *] \equiv \mathbf{0})$
[SConsentU1]	$a \triangleleft \mathsf{consent}(x).P \xrightarrow{a \triangleleft \mathsf{consent}(r)} P\{^r/_x\}$
[SConsentU2]	$\dfrac{P \xrightarrow{a \triangleleft \mathsf{consent}(r)} P'}{\{P\}_{\mathsf{id}} \xrightarrow{a \triangleleft \mathsf{consent}(r)@\mathsf{id}} \{P'\}_{\mathsf{id}}}$
[SConsentS1]	$a \triangleright \mathsf{consent}(r).P \xrightarrow{a \triangleright \mathsf{consent}(r)@\mathsf{id}} P \mid \overline{r} \triangleright [\mathsf{id} \otimes *]$
[SWithdraw1]	$r \triangleleft \mathsf{withdraw}.P \xrightarrow{r \triangleleft \mathsf{withdraw}} P$
[SWithdraw2]	$\dfrac{P \xrightarrow{r \triangleleft \mathsf{withdraw}} P'}{\{P\}_{\mathsf{id}} \xrightarrow{r \triangleleft \mathsf{withdraw}@\mathsf{id}} \{P'\}_{\mathsf{id}}}$
[SWithdraw3]	$\overline{r} \triangleright [\mathsf{id} \otimes c] \xrightarrow{\overline{r} \triangleright \mathsf{withdraw}@\mathsf{id}} \overline{r} \triangleright [\mathsf{id} \otimes *]$
[PId]	$\dfrac{P \xrightarrow{\ell} P' \quad \ell \neq a \triangleleft \mathsf{consent}(r), r \triangleleft \mathsf{withdraw}}{\{P\}_{\mathsf{id}} \xrightarrow{\ell} \{P'\}_{\mathsf{id}}}$
[SId]	$\dfrac{\{P\}_{\mathsf{id}} \xrightarrow{\ell} \{P\}_{\mathsf{id}}}{\mathsf{S}[\{P\}_{\mathsf{id}}] \xrightarrow{\ell} \mathsf{S}[\{P\}_{\mathsf{id}}]}$
[SOut2]	$\overline{r} \triangleright [\_ \otimes *] \xrightarrow{\overline{r}!\langle \_ \otimes * \rangle} \overline{r} \triangleright [\_ \otimes *]$
[SInp2]	$\overline{r} \triangleright [\_ \otimes *] \xrightarrow{\overline{r}?(\mathsf{id} \otimes c)} \overline{r} \triangleright [\_ \otimes *]$

is created on reference r with identity id. Following the dual definition of labels, a process that gives consent interacts with a process that receives consent to create a new store and exchange the corresponding reference. Withdraw follows a similar fashion. A process with a withdraw prefix observes a withdraw label $r \triangleleft \mathsf{withdraw}$ (rule [SWithdraw1]), and it is lifted to a user process via label $r \triangleleft \mathsf{withdraw}@\mathsf{id}$ (rule [SWithdraw2]). Finally, rule [SWithdraw3] observes a store receiving a withdraw request via label $\overline{r} \triangleright \mathsf{withdraw}@\mathsf{id}$ and as a result it deletes the corresponding private data and assigns the anonymous identity and the empty value to its memory. The next two rules bridge the gap between the new process $\{P\}_{\mathsf{id}}$ and System $\mathsf{S}[\{P\}_{\mathsf{id}}]$ and the rest of the calculus. Finally, two new rules define the interaction of the empty store.

5.2 Typing GDPR Compliance

As we have already discussed, the objective of this work is to propose a formal framework for modeling software systems and associated analysis techniques for guaranteeing that systems comply to privacy-related requirements. In this section, we discuss such an analysis technique and we show that the intended requirements we have discussed regarding the provision and withdrawal of consent can be statically checked by typechecking.

For instance, consider the following system:

$$\mathsf{User}[a!\langle \mathsf{id} \otimes c \rangle.P] \parallel \mathsf{Sys}[a?(x \otimes y).Q]$$

This system describes the situation, where private data are sent (resp. received) via a non-reference channel. However, this leads to the dissemination and processing of private data without proper permission and consent and, while it is a legal process, it clearly fails to satisfy the GRPR requirements. Note that exchange of private data can instead be achieved by communicating the reference to the associated private data store. A similar error would have occurred, if upon acquisition of private data from a store, a subcomponent of a system created a new store and thereby copied the data. However, this would lead to a situation, where the withdrawal of consent by the user would not reach the newly-defined store and thus, the system would continue to hold the data violating the right of the user for her data to be forgotten. This suggests that copying of private data should (if carried out) be done with care and ensuring that a link is maintained between the various copies of user data.
In the sequel, we propose an approach for recognizing these and other undesirable situations via typechecking, and we prove that well-typed processes do not present error, such as the ones discussed above. We

Table 4: Typing Rules.

Rule	Structure
[TCons]	$\dfrac{\Gamma,x{:}\mathsf{G}[\mathsf{t}[\mathsf{g}]];\Lambda \vdash P \quad \Gamma \vdash u{:}\mathsf{G}'[\mathsf{G}[\mathsf{t}[\mathsf{g}]]]}{\Gamma;\Lambda \vdash u \triangleleft \mathsf{consent}(x).P}$
[TStore]	$\dfrac{\Gamma,r{:}\mathsf{G}[\mathsf{t}[\mathsf{g}]];\Lambda \vdash P \quad \Gamma \vdash u{:}\mathsf{G}'[\mathsf{G}[\mathsf{t}[\mathsf{g}]]]}{\Gamma;\Lambda,\overline{r} \vdash u \triangleright \mathsf{consent}(r).P}$
[TWithdraw]	$\dfrac{\Gamma;\Lambda \vdash P \quad \Gamma \vdash u{:}\mathsf{G}[\mathsf{t}[\mathsf{g}]]}{\Gamma;\Lambda \vdash u \triangleleft \mathsf{withdraw}.P}$
[TId]	$\dfrac{\Gamma;\Lambda \vdash P}{\Gamma;\Lambda,\mathsf{id} \vdash \{P\}_{\mathsf{id}}}$

first define the set of types that are used to characterize the channels and the values that are used by the typing system. A type, T, is of the following form:

$$\begin{array}{lcl} \mathsf{g} & ::= & \mathsf{nat} \mid \mathsf{bool} \mid (\mathsf{g}_1,\dots,\mathsf{g}_n) \mid \dots \\ T & ::= & \mathsf{t}[\mathsf{g}] \mid \mathsf{G}[T] \mid \mathsf{g} \end{array}$$

A type g is a primitive data type, e.g. natural numbers and boolean, or a structure of primitive data $(\mathsf{g}_1,\dots,\mathsf{g}_n)$. A type $\mathsf{t}[\mathsf{g}]$ is used to type private data values $\mathsf{id} \otimes c$, where constant c is of type g. Channels are typed the $\mathsf{G}[T]$ type, that denotes a channel that can only send channels or values of type T between participants beloning to group G.

Our type system extends the one of the Privacy calculus by the rules in Table 4. It is based on the notion of a typing context defined as follows:

$$\Gamma \ ::= \ \Gamma, t:T \mid \emptyset \qquad \Lambda \ ::= \ \Lambda,\overline{r} \mid \Lambda,\mathsf{id} \mid \emptyset$$

A typing context Γ, or shared typing environment, maps values and variables to types. Operator Γ_1,Γ_2 is defined as $\Gamma_1 \cup \Gamma_2$. A typing context Λ, or linear typing environment, includes store references $\overline{r}$ and identifiers id. It is used to track names in a system and ensure that store references and identities are unique in a system. Operator Λ_1,Λ_2 is defined as $\Lambda_1 \cup \Lambda_2$, whenever $\Lambda_1 \cup \Lambda_2 = \emptyset$ and undefined otherwise. Typing judgments are of the following form:

$$\Gamma \vdash t:T \qquad \Gamma;\Lambda \vdash P \qquad \Gamma;\Lambda \vdash S$$

The first judgment states the a meta-variable t has a type T following a typing context. The second judgment states that a process P is well typed given in the typing contexts Γ and Λ. Similarly for the typing judgment for systems.

The first typing rule in Table 4 checks for the correct usage of the $u \triangleleft \mathsf{consent}(x).P$ term. It first checks that process P is correctly typed under a typing environment. It then checks whether the type of variable x can carry private data, i.e. it is of store reference type. Moreover, it checks that the type of channel u can carry an x variable type. In the premise of the rule variable x is not present because x is bounded in process P. Rule [TStore] follows a similar logic to the previous rule. However, it is checked that name r does not already have a store present in P by adding r in environment Λ. If r was already present in Λ it means that a store (or a consent for a store) is already present in process P. Rule [TWithdraw] performs a check that a withdraw can be performed on a channel that carries private data, i.e. it is a reference channel. The final rule, rule [TId], types a process which is equipped with an identity. The rule tracks the identity in the Λ environment. The presence of the identity in the Λ environment ensures that it cannot be used by another process as an identity in the system, because Λ cannot be composed with another linear environment that contains the same identity.

A main property of the system is that the execution of computations steps by a statically-typed system, yields systems that are also well typed.

Theorem 5.1 (Type Preservation)**.** *Let S be a system such that $\Gamma;\Lambda \vdash S$ for some Γ and Λ. Whenever $S \longrightarrow S'$ then there exists Γ',Λ' such that $\Gamma';\Lambda' \vdash S'$.*

Type preservation is an important theorem that states the soundness of the typing system. Moreover, as we discussed above, the cases of mishandling private data are not allowed in statically checked processes. This is formally given by the a type safety theorem. First, we define the notion of the error process, which are processes that mishandle private data.

Definition 5.1 (Error Process)**.** *A process P is an error process whenever it is of one of the forms:*

$$\begin{array}{c} \overline{r} \triangleright [\iota \otimes \delta] \mid \overline{r} \triangleright [\iota' \otimes \delta'], \\ a \triangleright \mathsf{consent}(x).P \ \textit{if}\ \overline{r} \in \mathsf{fn}(P), \\ a!\langle \mathsf{id} \otimes c \rangle.P, \qquad a?(x \otimes y).P, \\ r!\langle v \rangle.P, \ \textit{and} \qquad r?(x).P \end{array}$$

A system is an error system whenever, either it is of the form $\mathsf{G}[P]$ and $P \equiv (\nu\, \tilde{n})(P_1 \mid \dots \mid P_n)$ and $\exists i, 1 \leq i \leq n$ such that P_i is an error processes; or of the form $\mathsf{G}[S]$ and S is an error system; or of the form $(\nu\, \tilde{n})(S_1 \parallel \dots \parallel S_n)$ and $\exists i, 1 \leq i \leq n$ such that S_i is an error system.

The class of error systems captures cases where a system mishandles private data. The first case is where a system defines more than one stores on the same reference. This can lead to situations where the two stores have inconsistent private data.

The next case defines the situation, where one store is created with consent and another without consent. For example, in the system:

$$\mathsf{Sys}[a \triangleright \mathsf{consent}(r).(P \mid \overline{r} \triangleright [\mathsf{id} \otimes c])]$$

a second store will be created without consent.

The next two cases define the situation, where private data are sent (resp. received) via a non-reference channel with the possibility of leading to data processing without consent from the user.

The final two cases characterized as errors is when non private data are sent (resp. received) via a reference, e.g. the system

$$\mathsf{User}[r!\langle a\rangle.P] \parallel \mathsf{Sys}[\overline{r} \triangleright [\mathsf{id} \otimes c]]$$

cannot observe any meaningful interaction on reference r.

Based on a sound type system we can statically infer that a system is safe with respect to handling private data, i.e. a statically-checked system will never reduce to an error.

Theorem 5.2 (Type Safety)**.** *Let system S be a system such that $\Gamma \vdash S$ for some Γ. Whenever $S \longrightarrow^* S'$ then S' is not an error system.*

The proof follows from the fact that error systems cannot be successfully type checked and Theorem 5.1, that states that reduction preserves the typing property.

6 CONCLUSIONS

In this paper, we have extended the Privacy calculus of (Kouzapas and Philippou, 2017) with features to support the granting and withdrawal of consent by users, and for checking associated requirements as enunciated by the GDPR. While we have not considered privacy policies as in (Kouzapas and Philippou, 2017), where type checking is performed to ensure satisfaction of requirements pertaining to the purpose of usage, as well as the permissions endowed to each user in the system, we believe that the two approaches can easily be integrated being orthogonal to each other. Our vision is that a π-calculus-based framework can be developed and used by software engineers to model systems during the design phase, as well as in order to analyze and validate privacy-related properties, such as the GDPR-based provisions of *Lawfulness of Processing*, and the *Right to Erasure and Consent Withdrawal*. Analyzing and validating a system's specifications design model is a required step of software system creation. The modeling scheme introduced will provide information on conformance or may reveal any infringements found. We point out however, that this process cannot guarantee that the software engineers will implement the system to precisely conform to its specifications design, unless MDE techniques relying on the Privacy calculus are employed.

As future work, we intend to embed more GDPR provisions into the Privacy calculus. We will also work towards offering tools for easily expressing a system's specifications model in π-calculus-based formalism. Furthermore, tools for static checking actual code implementations, and tools for formally translating verified models to verified developed systems can be created, thus assisting the transition from the design phase to the development phase, by exploiting the ability of type-checking techniques at the coding level. Relevant work, that does not capture privacy requirements, but was developed in the context of the π-calculus and applies automated static analysis techniques to software code can be found in (Yoshida et al., 2013; Ng et al., 2015). Moreover, in future work we will focus on the notion of *purpose* by describing and validating purpose-based properties using a purpose-based, customizable privacy policy language, providing also a mechanism for privacy policy validation as embedded in the Privacy calculus. Our ultimate goal is to provide a holistic approach to privacy management in software system development.

REFERENCES

Abadi, M. and Gordon, A. D. (1997). A calculus for cryptographic protocols: The spi calculus. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 36–47. ACM.

Ahmadian, A. S., Strüber, D., Riediger, V., and Jürjens, J. (2018). Supporting privacy impact assessment by model-based privacy analysis. In *ACM Symposium on Applied Computing*, pages 1142–1149.

Backes, M., Hritcu, C., and Maffei, M. (2008). Type-checking zero-knowledge. In *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008*, pages 357–370.

Basso, T., Montecchi, L., Moraes, R., Jino, M., and Bondavalli, A. (2015). Towards a uml profile for privacy-aware applications. In *Proceedings of the IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM, 2015)*, pages 371–378. IEEE.

Bengtson, J., Bhargavan, K., Fournet, C., Gordon, A. D., and Maffeis, S. (2011). Refinement types for secure implementations. *ACM Transactions on Programming Languages and Systems*, 33(2):8.

Boussetoua, R., Bennoui, H., Chaoui, A., Khalfaoui, K., and Kerkouche, E. (2015). An automatic approach to transform bpmn models to pi-calculus. In *Proceedings of the International Conference of Computer Systems and Applications (AICCSA, 2015)*, pages 1–8. IEEE.

Braghin, C., Gorla, D., and Sassone, V. (2006). Role-based access control for a distributed calculus. *Journal of Computer Security*, 14(2):113–155.

Bugliesi, M., Colazzo, D., Crafa, S., and Macedonio, D. (2009). A type system for discretionary access control. *Mathematical Structures in Computer Science*, 19(4):839–875.

Cardelli, L., Ghelli, G., and Gordon, A. D. (2000). Secrecy and group creation. In *International Conference on Concurrency Theory*, pages 365–379. Springer.

Cavoukian, A. (2008). Privacy by design. Information Commissioner's Office.

Compagnoni, A. B., Gunter, E. L., and Bidinger, P. (2008). Role-based access control for boxed ambients. *Theoretical Computer Science*, 398(1-3):203–216.

Data Protection and Privacy Commissioners (2010). Resolution on privacy by design. In Proceedings of ICDPPC'10.

Dezani-Ciancaglini, M., Ghilezan, S., Jaksic, S., and Pantovic, J. (2010). Types for role-based access control of dynamic web data. In *Proceedings of WFLP'10*, LNCS 6559, pages 1–29. Springer.

European Parliament and Council of the European Union (2015). General data protection regulation. Official Journal of the European Union.

Fournet, C., Gordon, A., and Maffeis, S. (2007). A type discipline for authorization in distributed systems. In *20th IEEE Computer Security Foundations Symposium, CSF 2007, 6-8 July 2007, Venice, Italy*, pages 31–48.

Fowler, M. (2004). *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional.

Gjermundrød, H., Dionysiou, I., and Costa, K. (2016). privacytracker: A privacy-by-design gdpr-compliant framework with verifiable data traceability controls. In *Proceedings of the International Conference on Web Engineering*, pages 3–15. Springer.

Havey, M. (2005). *Essential business process modeling*. " O'Reilly Media, Inc.".

Hennessy, M. (2007). *A distributed Pi-calculus*. Cambridge University Press.

Hennessy, M., Rathke, J., and Yoshida, N. (2005). safedpi: a language for controlling mobile code. *Acta Informatica*, 42(4-5):227–290.

Hennessy, M. and Riely, J. (2002). Resource access control in systems of mobile agents. *Information and Computation*, 173(1):82–120.

Hes, R. and Borking, J. (1998). Privacy enhancing technologies: the path to anonymity. *ISBN*, 90(74087):12.

Hintze, M. and LaFever, G. (2017). Meeting upcoming gdpr requirements while maximizing the full value of data analytics.

Hustinx, P. (2010). Privacy by design: delivering the promises. *Identity in the Information Society*, 3(2):253–255.

Jürjens, J. (2002). Umlsec: Extending uml for secure systems development. In *Proceedings of the International Conference on The Unified Modeling Language*, pages 412–425. Springer.

Kalloniatis, C., Kavakli, E., and Gritzalis, S. (2008). Addressing privacy requirements in system design: the pris method. *Requirements Engineering*, 13(3):241–255.

Kapitsaki, G., Ioannou, J., Cardoso, J., and Pedrinaci, C. (2018). Linked usdl privacy: Describing privacy policies for services. In *2018 IEEE International Conference on Web Services (ICWS)*, pages 50–57. IEEE.

Kapitsaki, G. M., Kateros, D. A., Pappas, C. A., Tselikas, N. D., and Venieris, I. S. (2008). Model-driven development of composite web applications. In *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, pages 399–402. ACM.

Kapitsaki, G. M. and Venieris, I. S. (2008). Pcp: privacy-aware context profile towards context-aware application development. In *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, pages 104–110. ACM.

Kouzapas, D. and Philippou, A. (2017). Privacy by typing in the π-calculus. *Logical Methods in Computer Science*, 13(4).

Lam, V. S. (2008). On π-calculus semantics as a formal basis for uml activity diagrams. *Prooceedings of the International Journal of Software Engineering and Knowledge Engineering*, 18(04):541–567.

Milner, R., Parrow, J., and Walker, D. (1992). A calculus of mobile processes, parts I and II. *Information and Computation*, 100(1):1–77.

Ng, N., de Figueiredo Coutinho, J. G., and Yoshida, N. (2015). Protocols by default - safe MPI code generation based on session types. In *Proceedings of International Conference on Compiler Construction, CC 2015*, pages 212–232.

Perera, C., McCormick, C., Bandara, A. K., Price, B. A., and Nuseibeh, B. (2016). Privacy-by-design framework for assessing internet of things applications and platforms. In *Proceedings of the 6th International Conference on the Internet of Things*, pages 83–92. ACM.

Rubinstein, I. S. (2011). Regulating privacy by design. *Berkeley Technology Law Journal*, 26:1409.

Sangiorgi, D. and Walker, D. (2003). *The pi-calculus: a Theory of Mobile Processes*. Cambridge University Press.

Schmidt, D. C. (2006). Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2):25.

Thatte, S. (2001). Xlang: Web services for business process design. *Microsoft Corporation*, 2001.

Yoshida, N., Hu, R., Neykova, R., and Ng, N. (2013). The Scribble protocol language. In *Proceedings of TGC 2013, Revised Selected Papers*, pages 22–41.