International Journal of Software Engineering
and Knowledge Engineering
Vol. 29, No. 10 (2019) 1557–1604

DOI: 10.1142/S0218194019500499

World Scientific
www.worldscientific.com

A Privacy Awareness System for Software Design

Inah Omoronyia*, Ubong Etuk† and Peter Inglis‡

School of Computing Science
University of Glasgow
Glasgow G12 8QQ, UK
**inah.omoronyia@glasgow.ac.uk*
†*u.etuk.1@research.gla.ac.uk*
‡*p.inglis.1@research.gla.ac.uk*

Received 14 June 2019
Revised 2 September 2019
Accepted 17 September 2019

There have been concerting policy and legal initiatives to mitigate the privacy harm resulting from badly designed software technology. But one main challenge to realizing these initiatives is the difficulty in translating proposed principles and regulations into concrete and verifiable evidence in technology. This is partly due to the lack of systematic techniques and tools to address privacy in the software design, hence making it difficult for the designer to measure disclosure risk in a more intuitive way, taking into account the privacy objective that matters to each end user. To bridge this gap, we propose a framework for verifying the satisfaction of user privacy objectives in software design. Our approach is based on the (un)awareness that users acquire when information is disclosed, as it relates to the communication properties of objects in a design. This property is used to determine the expected privacy utility that users will derive from the design for a specified privacy objective. We demonstrate through case studies how this approach can help designers determine which design decision undermines users' privacy expectations and better design alternatives.

Keywords: Software design; privacy engineering; awareness.

1. Introduction

Seemingly innocuous design decisions during software engineering can unintentionally affect user privacy. This is aggravated with ubiquitous systems such as wearables, cars and services that we depend on now becoming privacy threats because of their ability to seamlessly communicate with each other [25]. Bad software design

* Corresponding author.

may undermine information protection by distorting user expectations and obscuring privacy harm. When this happens, users are tempted to give up on privacy and lose confidence in the technology. For example, studies have shown that most adults do not believe online service providers will keep their data private and secure [39].

Whereas it is common for software engineers to design the systems and not consider privacy, but rather delineated as the end users' problem for not being able to exercise control. Even worse is where privacy is only considered as an afterthought later in the development life cycle. In part, this is due to lack of incentives to invest in secure design since end users cannot readily tell which software preserves privacy and which does not at the point of sale [2]. This can make the designers comfortable ignoring or indulging in privacy-corrosive design and hoping for the best [25]. A more savory reason is that designers often lack appropriate tools and analysis techniques to determine the extent to which a design representation preserves privacy [40, 41]. Irrespective of the rationale for ignoring privacy, it demonstrates the impact of the lack of systematic consideration of privacy during software design.

It is also a common practice to align users' privacy needs and regulatory compliance through *privacy policies* — describing data collection, processing and distribution activities carried out by the software. The specification of such policies is majorly business–driven and carried out by legal experts with little or no insights from the software engineer. This makes it difficult to link policy statements with demonstrable evidence of their satisfaction in software design [4]. A further undesirable observation is that these statements have become more complicated over the years and now are rarely read by users. For example, Google's privacy policy statement has grown from 600 to 4000 words over the past 20 years, and their adjustments in response to GDPR have seen a 30% increase in the number of words.[a] The use of policy-based solutions in this way is symptomatic of masking deeply rooted privacy design problems in software with superficial privacy policy statements as solutions.

Consequently, the *privacy by design* paradigm has become a vital part of the dialog on what counts as good and effective privacy, and is often used as a slogan for systems built with privacy considered from the onset. Its opposite is responding to privacy harm after it has occurred. This initiative was passed by the Privacy Commissioners and Data Protection Authorities as an essential component of fundamental privacy protection. The objective was to help companies protect privacy by embedding a set of seven foundational principles into the design specification of technologies, business practices and physical infrastructures [14]. One of the principles even argues for the need to support privacy promises with a systematic and demonstrable evidence.

But the challenge with privacy by design is the lack of any underpinning or engineering understanding of those principles during software design [15, 25]. This research is therefore motivated by the increased need for demonstrable evidence of privacy preservation in software design. The expectation is that such evidence will

[a] https://policies.google.com/privacy/archive?hl=en-US.

help software designers make appropriate design choices. Specifically, this paper investigates a framework for demonstrating that interactions between objects in a software design enhance or abate the ability to realize a privacy objective. Such evidence can then be used by designers to distinguish between a good and a bad design from a privacy standpoint. This research is timely. Given the multitude of design patterns used in designing software, it remains unclear how the interaction between objects impacts on the ability of the software to preserve privacy; in particular, service-oriented software used for "wiring up" everyday objects to be part of the Internet of Things, social networks, mobile systems and e-commerce.

The central thesis of this research is that a software design that preserves privacy uses the appropriate *disclosure protocol* to enable interaction between objects. Such disclosure protocol maximizes the satisfaction of a privacy objective. The framework for investigating this assumption in a software design is shown in Fig. 1.

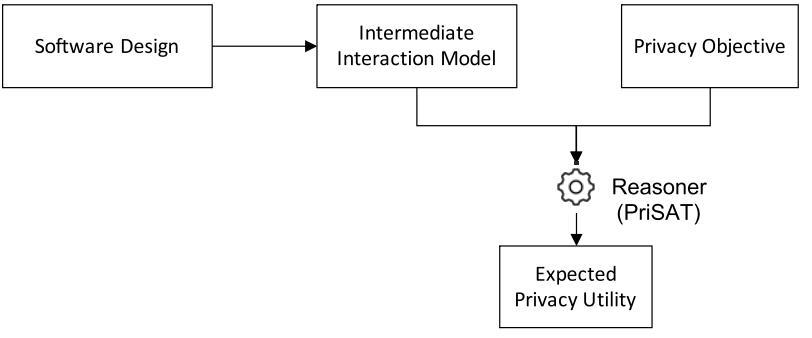


Fig. 1. Framework for verifying the satisfaction of a privacy objective in a software design.

The framework takes as input a software design representing the behavior of a system. This input is used to generate an intermediate interaction model (Secs. 4–6). The model identifies interacting objects and provides insights on the interaction history that the design will generate when implemented. In Sec. 4.1, we demonstrate how this history differs from the perspective of each object, and how it evolves based on the dynamic assignment of roles as information subject, sender or recipient. Differing interaction history implies that the (un)awareness of interacting parties (and therefore the extent of privacy realized) also differs. The interaction model represents this distinct (un)awareness in the memory of objects in Sec. 4.2. We demonstrate how a spectrum of (un)awareness ranging from being fully unaware to fully aware can be measured. When actions specified in a design are invoked, there is a transformation in the memory of associated objects along this spectrum. From an analytical viewpoint, these actions are synonymous to information-flow transactions defined by an object either requesting, consenting, sending or notifying another object about information. In Sec. 5, we outline how these memory transformations occur. Typically, information disclosure will consist of an ordered set of one or more

transactions to define a disclosure protocol. Section 6 outlines precedence rules for transactions in a disclosure protocol, and also the consistency of object memory after transformation.

We implemented PriSAT[b] as a tool for reasoning about privacy in software design. The tool generates an intermediate interaction model representing a software design and facilitates the mapping of disclosure protocols to interaction between objects specified in the design. Given a privacy objective and assigned disclosure protocols, PriSAT generates the expected privacy utility that each object will derive from the design. In this manner, the designer can compare different design alternatives to make informed design choice.

Our evaluation consists of two case studies. In Sec. 7.2, a followership network design on Twitter is reverse engineered. The case study focuses on the scenario where users interact with their followers privately. Our choice is based on the view that Twitter represents an example of a system where the traditional assumption that the impact of privacy can be localized to avoid contagion on other users' privacy becomes difficult to hold [58]. This is mainly due to the temporal and spatial distributions of objects, as well as the autonomous nature of associated users. On Twitter, it is easy for information once disclosed to reach unintended recipients, and users may be unsure if an information-flow path will ultimately lead to privacy violation. The study showed that the design's ability to realize a privacy objective on Twitter varied as information flowed in the network. This suggests that using the same disclosure protocol to foster interaction between any objects introduced asymmetry in the level of privacy each gets on the network. It was also observed that there is a maximum limit on the level of privacy that a design can offer. We demonstrated how a lightweight refactoring of the design can improve this limit.

The second case study in Sec. 7.3 presents a scenario analysis involving a family of interaction patterns for designing service-oriented software systems. These ranged from patterns where interactions between the information producer and consumer are facilitated or mediated via a broker using a push or pull mechanism, or a hybrid of both, with or without binding. Popular frameworks and standard specifications such as Message Queuing Telemetry Transport (MQTT) [18] for facilitating interaction between objects in Internet of Things, RESTful APIs [31] for e-commerce applications and messaging systems [33] are designed using one or more of these patterns. Using scenario analysis, our aim is to highlight the limits of the privacy-preserving capabilities of these patterns. This study demonstrates how design choices in a service-oriented software system inhibit or enhance the ability for an information producer, broker and/or consumer to realize a privacy objective. We show how the choice of a design pattern can be determined based on a balance of functional and privacy expectations. On the whole, this research will benefit academia, industry and policy makers with interest in addressing privacy by design challenges of real-world software systems.

[b]http://www.prisat.org.

2. Background

Software design and privacy are the core concepts of this research. There are numerous views on what constitutes a good software design [25, 48, 56, 12]. Broadly, software design comprises the specification of how systems are architected, how it functions, how it communicates and how the architecture, function and communication affect users. A design is judged to be good if it leads to software that is correct (does what it should), robust (tolerant of misuse), flexible (adaptable to shifting requirements), reusable, efficient, reliable and usable [16, 22]. Ultimately, a good design depends on the software engineer making the right design decisions. This is important since every design decision reflects an intent on how the software is to function or be used, as well as users' expectations as to how the software is compatible with contextual norms.

Likewise, there are diverse views of privacy in software. But we focus on a viewpoint that is based on (un)awareness, as it relates to communication properties of objects in a design and how end users interact via the objects. Such properties imbibe object states in the design which directly impacts the ability of the software to preserve privacy. This is as a consequence of the information disclosed to users when an object enters or exits a state. Broadly, (un)awareness is central to the manner by which we foster interactions in social settings. We coordinate and regulate our disclosure behavior based on our awareness about those we interact with and what information we want them to be aware or unaware. In a software-mediated setting, if a user was previously unaware of a fact, then a subsequent disclosure action may evolve the *memory* of objects representing the user, making the user aware of that fact. It is assumed here that the (un)awareness that is modeled in an object is perceived in the same way as its user. Hence, the two factors that can influence a user's disclosure behavior are: (1) the user's current (un)awareness; and (2) the desired (un)awareness that other users should have after disclosure. Hence, privacy is the ability for a user to regulate the evolution of its (un)awareness, and that of others, during information disclosure. The privacy objective during such regulation may be to increase awareness (information visibility) or unawareness (information secrecy) for one or more users.

Hence, verifying that a software design preserves privacy centers on how objects interact in the design and whether such interaction satisfies a privacy objective. We are of the view that a systematic approach to considering the disclosure protocols used for interaction between objects can (1) provide insights on the extent a privacy objective is satisfied and (2) be used to select a better design from a set of alternatives. There are many of such disclosure protocols. For example, in order to satisfy a privacy objective, it may be essential that when information is requested by a recipient, then the sender is required to seek consent from the subject before the information is sent to the recipient. For other cases, it may be that a granted consent is acknowledged by the sender, and the subject is notified when information is sent to the recipient [38]. Indeed, there are numerous ways to combine information request,

consent, send and notice actions to define a disclosure protocol, with each combination likely to generate a different level of privacy satisfaction since it results in a unique (un)awareness transformation in the memory of associated objects.

3. Related Works

Hoepman [29] defined a design strategy as an approach to achieve a certain design goal. This favors certain structural organization of the design or schemes over others, and contains properties that enable its distinction from other approaches that achieve the same goal. Hoepman also extended this view to define a privacy design strategy as a design strategy that achieves (some level of) privacy protection as its goal. We leverage on this view to investigate how the structure of a design and the limitations on disclosure protocols can vary across software design, as well as provide insights into the effectiveness of an alternative.

This research contributes to achieving privacy by design in software. While this paradigm has gained traction in policy circles, its actual integration into the design of software remains an open research question [24, 5]. A review by Bernsmed [9] summarized different approaches to operationalize privacy by design into existing software engineering processes. These include the Information and Privacy Commissioner of Ontario industry report on operationalizing privacy by design as a guide to implementing strong privacy practices [15] and the OASIS Privacy Management Reference Model and Methodology (PMRM) [47] for software engineering teams to analyze the system from a privacy perspective and to help them identify necessary technical and process mechanisms that should be implemented to support privacy. Similarly, Microsoft [46] and NOKIA [45] have, respectively, presented their engineering methodologies to bridge the gap between privacy laws and principles and techniques to foster the realization of privacy by design. One novel contribution is in the area of privacy impact assessments [54]; specifically, to enable the designer to carry out an assessment of designed software platforms to determine the level of privacy risk that users are exposed to, and any associated mitigation measures.

Developer-centered security is an emerging research area focused on how to get developers to build more secure systems from the start [44, 57]. While the traditional focus of cybersecurity research has been on developing new technologies and systems, in recent years, this is shifting to understanding the software engineer and how they are supported in creating secure products [53, 23]. One central theme is to explore and improve the tool support and techniques that are available to software builders. While process-centric tools exist for understanding the relationship between the software engineering process and privacy [1, 35], there is very little research on how to provide the developer with insights on the privacy-preserving capability of the product itself. This research contributes to this endeavor by investigating an analysis technique that enables the developer to understand how their design approach to achieve information disclosure in software impacts on user privacy.

Normative research has been applied in reasoning about privacy. Breaux *et al.* [11] used description logic to analyze privacy in data-flow specification with multi-party expectations. Similarly, Barth *et al.* [7] proposed the use of a temporal logic framework for expressing and reasoning about normative protocols in privacy legislation. Calikli *et al.* [13] used inductive logic programming to learn privacy norms in social software. Furthermore, Aucher *et al.* [6] applied modal and deontic logic in reasoning about obligations, permissions, knowledge and information exchange in the context of privacy policy compliance. A similar technique is applied by He and Antón [26] in the modeling of privacy requirements in role engineering. Our reasoning mechanism complements these existing approaches. We applied an awareness model based on possible world semantics to understand the nature of interaction between interacting objects in a software design and the privacy implications.

Finally, this work relates to access control in computing which typically involves the need to divulge information to authorized *objects* only [8]. An object here is a generic term that refers to an active agent capable of initiating or performing a computation of some sort. Access modes are broadly categorized into read, write and execute privileges granted to an object. Our technique provides a means to investigate the underlying engineering actions that lead to granted privileges and associated privacy risk; for example, the manner in which information is requested, consent is sought after, information is sent and user is notified. Indeed, our proposed approach provides a mechanism to determine the extent to which the defined access control policies help preserve privacy.

4. Modeling Awareness, Unawareness and Privacy

The privacy threat envisaged relates to a networked setting where privacy is dependent not only on an individual's action but also on those of other users. If this interdependence is ignored during software design, it can lead to end-user perception of loss over the control of their personal information after disclosure. Our threat model therefore builds on the Communication Privacy Management Theory, which defines information disclosure management in terms of privacy ownership, control and turbulence [43]. This theory is based on the principle that users believe they own and have a right to control their private information, and such control is achieved using personal privacy rules. When others are given access to a user's private information, they become co-owners of that information. Such co-owners need to negotiate on the mutually agreeable privacy rules about telling others. Privacy boundary turbulence occurs when co-owners do not effectively negotiate and follow mutually-held privacy rules, subsequently providing the perception of control loss.

The focus of this paper is to mitigate the threat of privacy turbulence resulting from inappropriate information disclosure in a software design. Hence, an interaction model for analyzing software design is necessary to reveal the relationship between a privacy objective, the disclosure protocols used to enable interaction between objects

and the resulting privacy utility for end users. In this section, we discuss the components of this model.

4.1. *Behavior modeling with role-based interaction history*

General knowledge modeling involves creating a computer interpretable model of knowledge [34]. Adopting a similar approach, our aim is to create a model that represents the behavior exhibited when users disclose or receive information via their surrogate objects. This is a role-based interaction described by the information flow t_i, involving the disclosure of the proposition ψ about a subject (su) from a sender (s) to a recipient (r):

$$t_i(\psi) = (U_i, R), \tag{1}$$

where

- U_i is the set of objects associated with t_i;
- $R = \{\text{su}, s, r\}$ are the roles that object in U_i can assume;
- $\text{roles} : U_i \rightarrow 2^R$ is a function mapping each object u_j to a set of roles, $\text{roles}(u_j) \subseteq R$.

When the sender discloses ψ about itself to the recipient, then s and su refer to the same object. Alternatively, before ψ is disclosed by the sender to a recipient, it is either generated by the sender or is granted custody by the subject. In this case, s, su and r refer to different objects. Finally, an object can also assume the roles of subject and recipient. This is when the sender sends ψ about a subject to the subject. Here, su and r refer to the same object. The first and last scenarios highlight the duality of roles during information flow. The sender cannot send ψ to itself. This makes it impossible for s and r to reference the same object.

Furthermore, when ψ is disclosed by multiple objects during interaction, an information flow path is formed. Formally, a path is a sequence of information flows represented by

$$t_{0<i\leq n}(\psi) = \sum_{i=1}^{n} (U_i, R),$$

where n is the path length and $U_i \subseteq U$. When $s \in \text{roles}(u_j)$ at t_1, then u_j is the path source. Whereas, if u_j is the recipient of ψ at t_1, then the information flow at t_2 is dependent on u_j to switching its role from recipient to a sender. This switching of roles propagates ψ from t_1 up to t_n. A backtrack occurs along a path when the sender at t_i discloses ψ to a sender at $t_{k<i}$; for instance, an interaction involving u_1 and u_2 where u_1 sends ψ to u_2, then u_2 sends back ψ to u_1. A path terminates at t_n when the intended destination of ψ is reached without backtracking. Alternatively, a path terminates at t_n when a backtrack occurs. At this point, a cycle is formed and terminated to prevent paths of infinite length over ψ.

Disclosed information may be attributed to one or more subjects. For the latter, attribution can be predefined statically, with all the subjects in ψ determined before

disclosure at t_1. Alternatively, attribution is determined dynamically as the information flows along a path. In this case, an object that assumes the role of a sender at t_{n-1} may become a subject at t_n. The role of a subject is permanent once assigned and cannot be switched over the history of a path. Whereas, since a path terminates once a backtrack is observed, an object can switch the roles of sender and recipient no more than once and twice, respectively, along a path. Finally, information may flow concurrently along any two paths $t^1_{0<i\leq n}$ and $t^2_{0<i\leq m}$. The result is an interaction network consisting of all information-flow paths:

$$T_{\mathtt{net}}(\psi) = \{t^1_{0<i\leq n}(\psi), t^2_{0<i\leq m}(\psi), \ldots, t^j_{0<i\leq k}(\psi)\}.$$

Assume an interaction between the set of objects u_1–u_5 over ψ about u_1. First, u_1 discloses ψ to u_2 and u_3. Subsequently, u_2 and u_3 disclose ψ to u_4 and u_5, respectively. Finally, u_4 sends back ψ to u_1 and u_2 and also discloses to u_5. The graph diagram, interaction network (represented as adjacency matrix), resulting information flow paths and role-based interaction histories for static and dynamic subject attributions are illustrated in Fig. 2. In extracting paths from the interaction network, the order in which the disclosure actions are executed is nonconsequential and the longest possible path that satisfies backtracking constraint is assumed. It can be observed that each object in the network generates a peculiar interaction history based on its roles during each information flow. An alternative information-flow setting may result in a different interaction history. Our objective is to understand how these variations in disclosure behavior can be used to articulate the level of (un)awareness that objects attain about disclosed information. Subsequently, we explore how such (un)awareness can be regulated in a software design to preserve privacy.

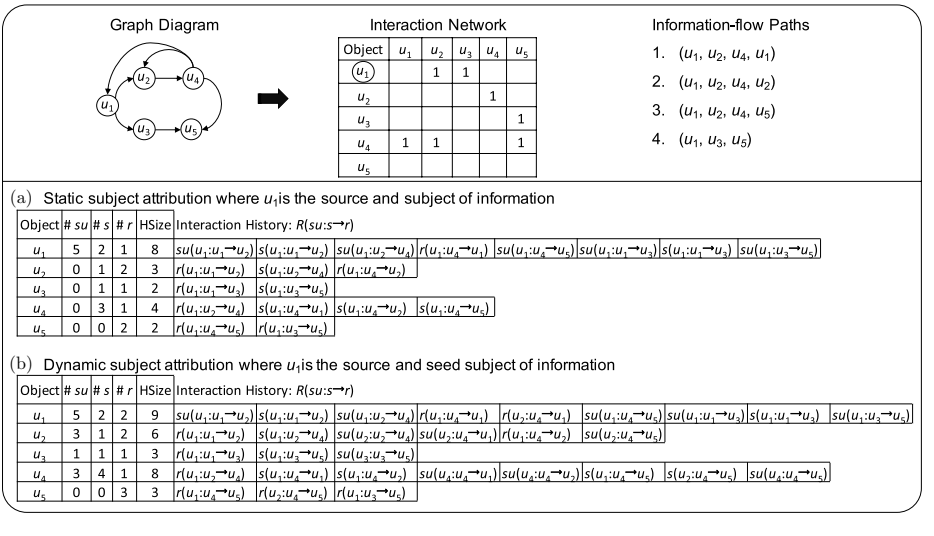


Fig. 2. Role-based interaction histories from the viewpoints of u_1–u_5 using static and dynamic subject attributions.

4.2. *The (un)awareness of objects*

Early works on reasoning about awareness in interactive settings assume limited rationality of objects [17, 20, 27]. This implies that when the information is disclosed, the awareness obtained by objects would vary within a spectrum of being fully aware of the information to fully unaware. In this Subsection, we leverage on this assumption to set the foundations for reasoning about the relationship between software design and the awareness generated by the interaction properties of objects in that design. To achieve this, the memory of an object based on its interaction with other objects about ψ is represented using possible world semantics for describing alternative worlds (modes) and accessibility relations between such worlds [49, 28]. When ψ is disclosed in an information flow, then the subject, sender and recipient who are previously unaware may become aware of ψ and its related propositions. Thus, the (un)awareness contained in the memory of an object about ψ can be modeled using $M(\psi) = (W, p, R, I, w_c)$, where

- W is a set of possible worlds each considering a unique viewpoint on ψ;
- p is referred to as the principal and represents the object whose memory contains the (un)awareness of ψ;
- $R \in 2^{\{\mathrm{su},s,r\}}, \forall \mathrm{su} \in \mathrm{SU}$, is a set of reference objects whose (un)awareness about ψ may be considered by the principal. Each subject, sender or recipient may assume the role of a principal and/or a reference object;
- $I \subseteq W \times W$ is the accessibility relation on W. Given two worlds $w_1, w_2 \in W$, the principal finds ψ in w_1 indistinguishable from ψ in w_2;
- w_c is the current world of p.

To distinguish between an object being aware and the one that is unaware, we define an awareness instance in the memory of an object to consist of a principal and the optional reference objects, represented by the modal operator A. Similarly, an unawareness instance contains a principal with optional reference objects, represented by the operator $\neg A$. A composite instance contains more than one operator. Otherwise, the instance is atomic. The utilization of this operator precludes the valuation of the truth of a proposition as expected in reasoning about knowledge. In the following, we generate four (un)awareness classes in the memory of p based on M.

[**A1**] The first class represents a principal being aware or unaware of the atomic proposition f (i.e. $\psi = f$). Instances of this class only contain the principal with no reference object, rendering R an empty set. Determining whether p is aware or unaware of f can be modeled by considering W as a set of two possible worlds. The first world is w_1 where p does not consider f possible and is written as $\neg f$. The second world, w_2, is the world where p considers f possible. The accessibility relations between these two worlds are illustrated in Fig. 3. From w_1, both w_1 and w_2 are accessible. Whereas, from w_2, only w_2 is accessible. The unshaded world are Fig. 3 represents the current world of p. When p's current world is w_1, then p is unaware of f and it is represented as $\neg A_p f$. This is because in this world, $\neg f$ in w_1 is

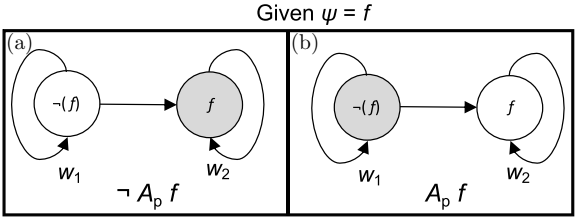


Fig. 3. The (un)awareness of a principal (p) about the proposition f.

indistinguishable from f in w_2. Conversely, when p's current world is w_2, then p is aware of f and it is represented as $A_p f$. This is because f is uniquely distinguishable from w_2 since it no longer considers w_1 accessible. In summary, let M_p represent the memory of principal p. When reasoning about p's (un)awareness of f, then $\neg A_p f$ and $A_p f$ exist in the memory of p thus

$$\mathbf{A1}_p = \{\neg A_p f, A_p f\} \quad \text{and} \quad \forall a \in \mathbf{A1}_p, a \in M_p.$$

[**A2**] The second class enables the principal to consider whether a reference object is aware or unaware of f. In this case, ψ is a composite proposition. Hence, **A2** is a class of (un)awareness that can be used to reason about instances in **A1**. Instances of **A2** can only contain one reference object. Thus, R may contain either the subject, sender or recipient. We refer to this potential reference object as r'. Determining whether p is aware or unaware that r' is aware or unaware of f can be realized by first generating the (un)awareness of r' about f as $\mathbf{A1}_{r'} = \{\neg A_{r'} f, A_{r'} f\}$. The next step is to consider every instance in $\mathbf{A1}_{r'}$ as ψ and apply similar heuristics used in realizing p's (un)awareness of f as highlighted in **A1** for each ψ. The result is a set of possible worlds, their accessibility relations and p's (un)awareness given its current world and ψ as shown in Fig. 4. In summary, when considering whether p is (un)aware that a reference object r' is (un)aware of f, then $\neg A_p A_{r'} f$, $A_p A_{r'} f$, $\neg A_p \neg A_{r'} f$ and $A_p \neg A_{r'} f$ exist in the memory of p thus

$$\mathbf{A2}_p = \{\neg A_p A_{r'} f, A_p A_{r'} f, \neg A_p \neg A_{r'} f, A_p \neg A_{r'} f\} \quad \text{and}$$
$$\forall a \in \mathbf{A2}_p, a \in M_p.$$

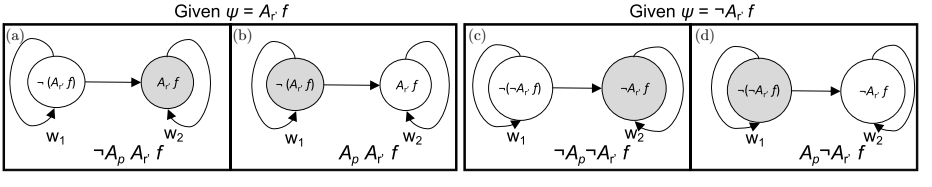


Fig. 4. The (un)awareness of a principal (p) about the (un)awareness of r'.

[**A3**] The third class enables the principal to consider the (un)awareness that a reference object may have about the principal. Hence, **A3** is also a composite

proposition and represents the class of (un)awareness that can be used to reason about instances in **A2**. Instances of **A3** contain one reference object and a principal. The reference can either be the subject, sender or recipient. Again, we refer to the potential first reference object as r'. Determining whether p is aware or unaware that r' is aware or unaware that p is aware or unaware of f is realized by first generating the (un)awareness of r' about p being aware or unaware of f. Thus

$$\mathbf{A2}_{r'} = \{\neg A_{r'} A_p f, A_{r'} A_p f, \neg A_{r'} \neg A_p f, A_{r'} \neg A_p f\}.$$

The next step is to consider every instance in $\mathbf{A2}_{r'}$ as ψ and apply similar heuristics used in realizing p's (un)awareness of f as highlighted in **A1** for each ψ. The result is a set of possible worlds, their accessibility relations and p's (un)awareness given its current world and ψ as shown in Fig. 5. In summary, when considering whether p is (un)aware that a reference object r' is (un)aware that p is (un)aware of f, then the memory of p is defined by adding every instance of $\mathbf{A3}_p$ to M_p. Thus

$$\begin{aligned}\mathbf{A3}_p = \{&\neg A_p \neg A_{r'} A_p f, A_p \neg A_{r'} A_p f, \neg A_p A_{r'} A_p f, A_p A_{r'} A_p f,\\ &\neg A_p \neg A_{r'} \neg A_p f, A_p \neg A_{r'} \neg A_p f, \neg A_p A_{r'} \neg A_p f, A_p A_{r'} \neg A_p f\} \quad \text{and}\\ &\forall a \in \mathbf{A3}_p, a \in M_p.\end{aligned}$$

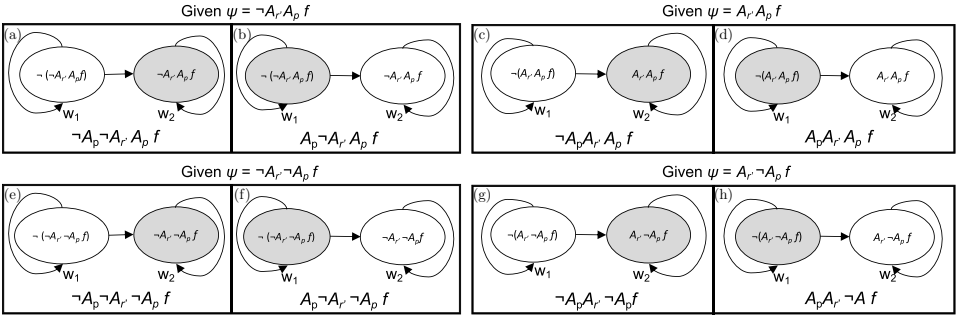


Fig. 5. The (un)awareness of a principal (p) about the (un)awareness of r' about the (un)awareness of p.

[**A4**] This final class enables the principal to consider the (un)awareness that a reference object has about other references. Hence, **A4** is also a class of (un)awareness that can be used to reason about instances in **A2** where ψ is a composite proposition. Instances of this class contain two reference objects, these are any ordered pair of subject, sender and recipient. We refer to this ordered pair as r_1 and r_2. Determining whether p is aware or unaware that r_1 is aware or unaware that r_2 is aware or unaware of f can be realized by first generating the (un)awareness of r_1 about r_2 being aware or unaware of f and *vice versa*. Thus

$$\begin{aligned}\mathbf{A2}_{r_1} &= \{\neg A_{r_1} A_{r_2} f, A_{r_1} A_{r_2} f, \neg A_{r_1} \neg A_{r_2} f, A_{r_1} \neg A_{r_2} f\},\\ \mathbf{A2}_{r_2} &= \{\neg A_{r_2} A_{r_1} f, A_{r_2} A_{r_1} f, \neg A_{r_2} \neg A_{r_1} f, A_{r_2} \neg A_{r_1} f\}.\end{aligned}$$

The next step is to consider every instance in $\mathbf{A2}_{r_1}$ and $\mathbf{A2}_{r_2}$ as ψ and apply similar heuristics used in realizing p's (un)awareness of f as highlighted in **A1** for each ψ. The result is a set of possible worlds, their accessibility relations and p's (un)awareness given its current world and ψ as shown in Fig. 6. In summary, when considering whether p is (un)aware that a reference object r_1 is (un)aware that another reference object r_2 is (un)aware of f, then the memory of p is defined by adding every instance of $\mathbf{A4}_p$ to M_p. Thus

$$\begin{aligned}\mathbf{A4}_p = \{&\neg A_p \neg A_{r_1} A_{r_2} f, A_p \neg A_{r_1} A_{r_2} f, \neg A_p A_{r_1} A_{r_2} f, A_p A_{r_1} A_{r_2} f,\\ &\neg A_p \neg A_{r_1} \neg A_{r_2} f, A_p \neg A_{r_1} \neg A_{r_2} f, \neg A_p A_{r_1} \neg A_{r_2} f, A_p A_{r_1} \neg A_{r_2} f,\\ &\neg A_p \neg A_{r_2} A_{r_1} f, A_p \neg A_{r_2} A_{r_1} f, \neg A_p A_{r_2} A_{r_1} f, A_p A_{r_2} A_{r_1} f,\\ &\neg A_p \neg A_{r_2} \neg A_{r_1} f, A_p \neg A_{r_2} \neg A_{r_1} f, z \neg A_p A_{r_2} \neg A_{r_1} f, A_p A_{r_2} \neg A_{r_1} f\}\\ &\text{and} \quad \forall a \in \mathbf{A4}_p, \ \ a \in M_p.\end{aligned}$$

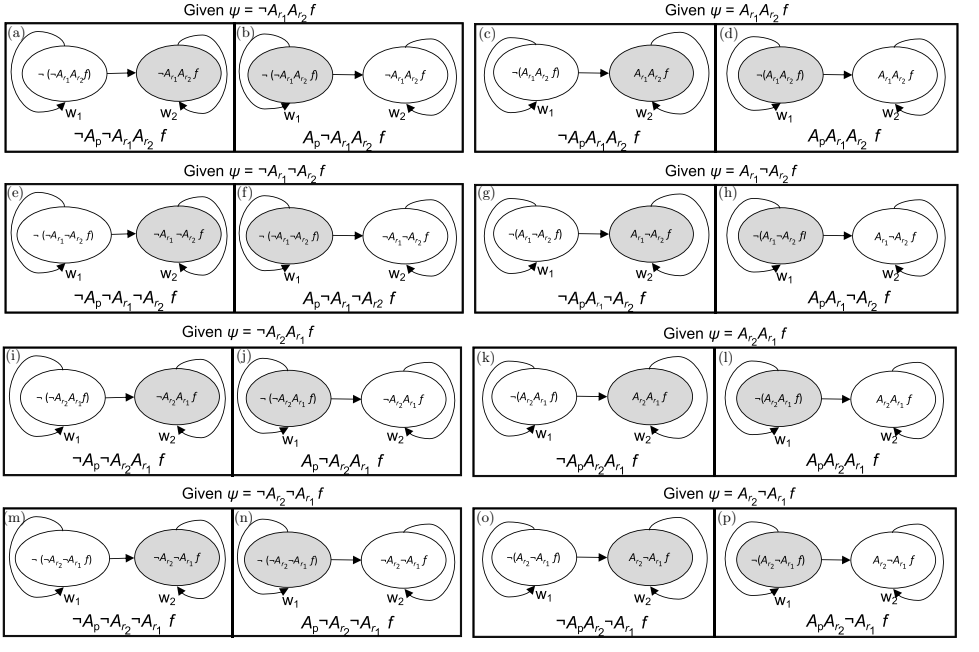


Fig. 6. The (un)awareness of a principal (p) about the (un)awareness of r_1 about the (un)awareness of r_2.

4.3. *Object memory*

The subject, sender or recipient can assume unique or dual roles as the principal and/or reference object in an instance of a class. When the roles are unique, then two consecutive (un)awareness operators cannot refer to the same object in that instance. For this case, there are 42 (un)awareness instances from **A1**, **A2**, **A3** and **A4**, respectively, in the memory of a principal as illustrated in Table 1. These instances

Table 1. The memory of objects involved in an information flow.

subject

	M_{su}			
A1	1	$A_{su}f$	2	$\neg A_{su}f$
A2	3	$A_{su}A_sf$	4	$\neg A_{su}A_sf$
	5	$A_{su}\neg A_sf$	6	$\neg A_{su}\neg A_sf$
	7	$A_{su}A_rf$	8	$\neg A_{su}A_rf$
	9	$A_{su}\neg A_rf$	10	$\neg A_{su}\neg A_rf$
A3	11	$A_{su}A_sA_{su}f$	12	$A_{su}A_s\neg A_{su}f$
	13	$A_{su}\neg A_sA_{su}f$	14	$A_{su}\neg A_s\neg A_{su}f$
	15	$\neg A_{su}A_sA_{su}f$	16	$\neg A_{su}A_s\neg A_{su}f$
	17	$\neg A_{su}\neg A_sA_{su}f$	18	$\neg A_{su}\neg A_s\neg A_{su}f$
	19	$A_{su}A_rA_{su}f$	20	$A_{su}A_r\neg A_{su}f$
	21	$A_{su}\neg A_rA_{su}f$	22	$A_{su}\neg A_r\neg A_{su}f$
	23	$\neg A_{su}A_rA_{su}f$	24	$\neg A_{su}A_r\neg A_{su}f$
	25	$\neg A_{su}\neg A_rA_{su}f$	26	$\neg A_{su}\neg A_r\neg A_{su}f$
A4	27	$A_{su}A_sA_rf$	28	$A_{su}A_s\neg A_rf$
	29	$A_{su}\neg A_sA_rf$	30	$A_{su}\neg A_s\neg A_rf$
	31	$\neg A_{su}A_sA_rf$	32	$\neg A_{su}A_s\neg A_rf$
	33	$\neg A_{su}\neg A_sA_rf$	34	$\neg A_{su}\neg A_s\neg A_rf$
	35	$A_{su}A_rA_sf$	36	$A_{su}A_r\neg A_sf$
	37	$A_{su}\neg A_rA_sf$	38	$A_{su}\neg A_r\neg A_sf$
	39	$\neg A_{su}A_rA_sf$	40	$\neg A_{su}A_r\neg A_sf$
	41	$\neg A_{su}\neg A_rA_sf$	42	$\neg A_{su}\neg A_r\neg A_sf$

sender

	M_s			
A1	1	A_sf	2	$\neg A_sf$
A2	3	$A_sA_{su}f$	4	$\neg A_sA_{su}f$
	5	$A_s\neg A_{su}f$	6	$\neg A_s\neg A_{su}f$
	7	A_sA_rf	8	$\neg A_sA_rf$
	9	$A_s\neg A_rf$	10	$\neg A_s\neg A_rf$
A3	11	$A_sA_{su}A_sf$	12	$A_sA_{su}\neg A_sf$
	13	$A_s\neg A_{su}A_sf$	14	$A_s\neg A_{su}\neg A_sf$
	15	$\neg A_sA_{su}A_sf$	16	$\neg A_sA_{su}\neg A_sf$
	17	$\neg A_s\neg A_{su}A_sf$	18	$\neg A_s\neg A_{su}\neg A_sf$
	19	$A_sA_rA_sf$	20	$A_sA_r\neg A_sf$
	21	$A_s\neg A_rA_sf$	22	$A_s\neg A_r\neg A_sf$
	23	$\neg A_sA_rA_sf$	24	$\neg A_sA_r\neg A_sf$
	25	$\neg A_s\neg A_rA_sf$	26	$\neg A_s\neg A_r\neg A_sf$
A4	27	$A_sA_{su}A_rf$	28	$A_sA_{su}\neg A_rf$
	29	$A_s\neg A_{su}A_rf$	30	$A_s\neg A_{su}\neg A_rf$
	31	$\neg A_sA_{su}A_rf$	32	$\neg A_sA_{su}\neg A_rf$
	33	$\neg A_s\neg A_{su}A_rf$	34	$\neg A_s\neg A_{su}\neg A_rf$
	35	$A_sA_rA_{su}f$	36	$A_sA_r\neg A_{su}f$
	37	$A_s\neg A_rA_{su}f$	38	$A_s\neg A_r\neg A_{su}f$
	39	$\neg A_sA_rA_{su}f$	40	$\neg A_sA_r\neg A_{su}f$
	41	$\neg A_s\neg A_rA_{su}f$	42	$\neg A_s\neg A_r\neg A_{su}f$

recipient

	M_r			
A1	1	A_rf	2	$\neg A_rf$
A2	3	$A_rA_{su}f$	4	$\neg A_rA_{su}f$
	5	$A_r\neg A_{su}f$	6	$\neg A_r\neg A_{su}f$
	7	A_rA_sf	8	$\neg A_rA_sf$
	9	$A_r\neg A_sf$	10	$\neg A_r\neg A_sf$
A3	11	$A_rA_{su}A_rf$	12	$A_rA_{su}\neg A_rf$
	13	$A_r\neg A_{su}A_rf$	14	$A_r\neg A_{su}\neg A_rf$
	15	$\neg A_rA_{su}A_rf$	16	$\neg A_rA_{su}\neg A_rf$
	17	$\neg A_r\neg A_{su}A_rf$	18	$\neg A_r\neg A_{su}\neg A_rf$
	19	$A_rA_sA_rf$	20	$A_rA_s\neg A_rf$
	21	$A_r\neg A_sA_rf$	22	$A_r\neg A_s\neg A_rf$
	23	$\neg A_rA_sA_rf$	24	$\neg A_rA_s\neg A_rf$
	25	$\neg A_r\neg A_sA_rf$	26	$\neg A_r\neg A_s\neg A_rf$
A4	27	$A_rA_{su}A_sf$	28	$A_rA_{su}\neg A_sf$
	29	$A_r\neg A_{su}A_sf$	30	$A_r\neg A_{su}\neg A_sf$
	31	$\neg A_rA_{su}A_sf$	32	$\neg A_rA_{su}\neg A_sf$
	33	$\neg A_r\neg A_{su}A_sf$	34	$\neg A_r\neg A_{su}\neg A_sf$
	35	$A_rA_sA_{su}f$	36	$A_rA_s\neg A_{su}f$
	37	$A_r\neg A_sA_{su}f$	38	$A_r\neg A_s\neg A_{su}f$
	39	$\neg A_rA_sA_{su}f$	40	$\neg A_rA_s\neg A_{su}f$
	41	$\neg A_r\neg A_sA_{su}f$	42	$\neg A_r\neg A_s\neg A_{su}f$

capture all the (un)awareness that a subject, sender or recipient will acquire about disclosed information and the related (un)awareness propositions. For each class, there is a set of (un)awareness instances with the same principal and reference object(s). For example, labels 3, 4, 5 and 6 in column M_{su} of Table 1 constitute the set of all (un)awareness instances of type **A2** in the memory of the subject with the sender as a reference object. Whereas, labels 7, 8, 9 and 10 are the set of all (un) awareness instances of type **A2** in the memory of the subject with the recipient as a reference object. This uniqueness of roles in an information flow represents the (un) awareness an object has about the (un)awareness of other parties during disclosure.

When roles are not unique, then a principal can also be a reference object. In this case, two consecutive (un)awareness operators may refer to the same object. This duality of roles during information flow depicts self-awareness where an object is (un)aware of its own (un)awareness. This phenomenon is normally assumed in standard models of belief and knowledge as positive and negative introspections [10, 55]. An object is positively introspective if it is aware that it is aware of ψ whenever it is aware of ψ. Similarly, an object is negatively introspective if it is aware that it is unaware of ψ whenever it is unaware of ψ. This observation does not affect the instances of **A1** since it only contains a principal. Whereas, for instances of **A2** and **A3**, the principal and the reference are the same object. Hence, instance labels 3, 7 and 4, 8 in **A2**, respectively, depict the ability/inability of an object to positively introspect given that the object becomes (un)aware that it is aware of f. Also, labels 5, 9 and 6, 10, respectively, represent the ability/inability of an object to negatively introspect given that the object is (un)aware that it is (un)aware of f. Similar views of positive and negative introspections are observed for labels 11–26 in **A3**, where an object is (un)aware that it is (un)aware that it is (un)aware of f. For **A4**, although the two reference objects remain a unique pair, the principal and one element of the pair can refer to the same object. This results in positive and negative introspections

for labels 47–42, where an object becomes (un)aware that it is (un)aware that another object is (un)aware of f.

4.4. *Awareness differential*

At any moment in time, an (un)awareness instance may or may not be tenable in the memory of an object. An instance becomes untenable when a principal considers it no longer reasonable as a result of a disclosure action, and vice versa. For example, it is tenable for the recipient to consider that it is aware of f after it receives f from the sender. Conversely, it is untenable for a recipient to consider that it is unaware of f after it has received f from the sender. The transition of an awareness instance from a tenable to untenable state or vice versa is based on the executed disclosure protocol during information flow as detailed in Sec. 6. Categorizing each (un)awareness instance in the memory of a principal in this manner enables the distinction between different levels of (un)awareness resulting from an information flow.

Definition 1. A principal p is *fully unaware* of a proposition ψ if for every tenable instance a with respect to ψ in the memory of p, the actual world of p is w_1.

Otherwise, information flow generates *full* or *partial awareness* of ψ whose severity is determined by the number of tenable instances in w_1 or w_2. This measure depicts the level of doubt that the principal may have about disclosed information, and computed as the *awareness differential*

$$\texttt{ADiff}(p, \mathbf{A}_i) = \frac{\Omega - 1}{\lambda - 1} + \beta,$$

where

- Ω is the number of tenable instances of class $\mathbf{A}_i$ with the same reference object(s) in M_p;
- λ is the number of instances of class $\mathbf{A}_i$ with the same reference object(s) in M_p;
- β is the number of tenable unawareness instances of class $\mathbf{A}_i$ with the same reference object(s) in M_p, and acts as a discriminator between full awareness and full unawareness.

Assume for the class **A2** that it is only tenable for the principal to consider that it is aware that r_1 is aware of f. Then the awareness differential for this class in the memory of the principal is zero since this is the only active instance from **A2** relating to r_1. Whereas, if in addition it is tenable for the principal to consider that it is aware that r_1 is unaware of f, then the awareness differential becomes 0.33. Alternatively, when it is only tenable that the principal considers that it is unaware that r_1 is aware of f and that it is unaware that r_1 is unaware of f, then the principal is fully unaware with respect to **A2** since for both tenable instances the actual world of the principal is w_1. In this case, the awareness differential of the principal with respect to **A2** is 2.3. Overall, the awareness differential leans towards zero when the principal becomes fully aware.

Definition 2. A principal p is *fully aware* of ψ if for every tenable instance a with respect to ψ in the memory of p, the actual world of p is w_2 and $\texttt{ADiff}(p, \mathbf{A}_i) = 0$, where $\mathbf{A}_i$ is the class of a.

For instance, when the sender discloses information about the subject to a recipient, the subject attains full awareness when only the instances labeled 1, 3, 7, 11, 19, 27 and 35 are tenable in M_{su}. This implies that the subject considers it tenable that it is aware of f and aware that both the sender and recipient are aware of f. It also considers it tenable that it is aware that the sender and recipient are aware of its awareness of f. In addition, the subject considers that it is tenable that it is aware that the sender is aware that the recipient is aware of f. Finally, the recipient is aware that the sender is aware of f. At full awareness, the awareness differential for su over all awareness classes is therefore

$$\sum_{i=1}^{4} \texttt{ADiff}(\text{su}, \mathbf{A}_i) = 0.$$

5. Memory Transformation

The advantage of modeling (un)awareness using possible world semantics is the ability to highlight how disclosure protocols transform the memory of objects as information flows from one object to another. To illustrate this, we represent the disclosure protocol that results in (un)awareness transformations using transactions. A transaction contains processes responsible for updating the memory of object on what it is (un)aware in certain states. These transactions are labeled information `Request`, `Consent`, `Sent` and `Notice`, respectively. Transactions and associated processes are executed in sequence to ensure that the memory of objects does not end up in invalid (un)awareness states. We assume at the beginning of a path, before an information flow occurs (t_0), that the subject, sender and recipient are fully unaware. Based on this assumption, tenable instances in the memory of the subject, sender and recipient in Table 1 are labels 2, 4, 6, 8, 10, 15–18, 23–26, 31–34 and 39–42 for M_{su}, M_s and M_r, respectively.

A disclosure protocol constitutes a sequence of transactions that enables the sender, subject and recipient of an information flow to gain a level of awareness ranging from being partially aware to fully aware after disclosure. The consequence of a disclosure protocol in the memory of a principal at t_1 after disclosure is two-faceted. First, one or more previously tenable (un)awareness instances of the same class and reference object(s) become untenable. Second, for the same class and reference object(s), one or more previously untenable (un)awareness instances become tenable. This section highlights these memory transformations for `Request`, `Consent`, `Sent` and `Notice` transactions.

5.1. *Request*

This transaction involves a sender and a recipient, and is required to give the sender an opportunity to consider if a recipient should gain awareness of information. This is representative of a setting where privacy is achieved by enabling an object to accept or reject another object's request to gain awareness of information. The semantics involve a recipient requesting for information about a subject from a sender with a `request` process. Tenable awareness instances in the memory of the recipient and sender as a result of this transaction are shown in Table A.1 of Appendix A. There are two scenarios where information can be requested by a recipient from a sender. First is where the sender generates requested information about the subject. In such a scenario, the sender is the custodian of information about the subject. The alternative scenario is where the sender is not in possession of the requested information. In this case, the request can only be fulfilled if the information is sought from the subject. For example, in an event-driven system, a component (recipient) may request event notifications about another component (subject) from the event broker (sender). At the time of request, the broker is unaware of such notifications from the subject. Hence, the transaction can only be fulfilled when notifications are subsequently pushed by the subject component to the broker.

5.2. *Consent*

This transaction involves the sender and subject, and the aim is for a sender to disclose information to a recipient only when granted permission to do so by the subject. Obtaining `Consent` may be implicit where initiating the operation acts as the willingness to disclose; otherwise it may be explicit, where the user is clearly presented with an option to agree or disagree (opt-in or opt-out) of the collection or disclosure of personal information [21]. `Consent` can be specific or generic. The former involves a sender seeking consent or the subject granting consent to disclose information to a specified recipient. Hence, the sender has identified the recipients for which it is seeking or granting consent. The subject is also informed or able to identify the recipients to which information will be sent at the time `Consent` transaction is executed. Conversely, generic `Consent` involves the sender seeking or the subject granting consent to disclose information to unknown recipients. The semantics are as follows: The sender first seeks consent from the subject to disclose information to a recipient with `seekConsent`. Subsequently, the sought consent may be granted by the subject with `grantConsent`. Tenable awareness instances in the memory of the subject and sender as a result of this transaction for specific consent are shown in Table A.4 of Appendix A.

A `seekConsent` process can also be executed in one of the two ways. In the first case, the sender is the custodian of information about the subject for which the disclosure consent is being sought. Hence, the sender is aware of information to be disclosed at the point `seekConsent` is executed. Alternatively, the subject is the source of information for which the consent is being sought. In this case, the subject becomes aware of information to be disclosed at the point `seekConsent` is executed.

After `grantConsent` is executed, it becomes tenable for the subject and sender to consider that they are aware of the information for which the consent is being granted. Hence, it is impossible for the subject to grant consent to disclose information that it is not aware. Also, it is impossible for the sender to remain unaware of the information it has been granted consent to disclose. Tenable instances of **A2**, **A3** and **A4** are also dependent on whether the sender generates the information for which the consent is being granted, or the subject is the source of the information. Finally, when the consent is not granted by the subject for the information generated by the sender, but rather it responds to the sender with an acknowledgment of sought consent, then it is tenable for the subject to consider that it is aware of the information for which the consent is denied. Hence, it is impossible for the subject to acknowledge sought information that it is not aware (cf. Sec. 5.5).

5.3. *Sent*

This transaction enables a recipient to gain awareness of information about a subject and involves a sender and a recipient. The semantics entail the sender disclosing information to the recipient with a `send` process. After the execution of this process, the recipient becomes aware of disclosed information with the corresponding transformation in tenable **A1**–**A4** instances in the memory of sender and recipient as shown in Table A.2 of Appendix A.

5.4. *Notice*

A `Notice` transaction is used by an object to notify other objects of its state of (un)awareness. There are primarily two types of notifications — prominent and discoverable [37]. Prominent notice is the one that is designed to catch the user's attention and the user to inspect the outcome of their privacy options and choices. Whereas, discoverable notice is the one that the user has to find. Irrespectively, they both aim to achieve privacy by ensuring transparency — referring to openness to users in the manner personal information is manipulated [3]. These include which information is sent, who the receiver is and where the information came from. To achieve transparency, it is then an obligation for each object to inform other parties involved in an information flow of its (un)awareness. Realizing full transparency across parties involved in an information flow requires six forms of `Notice` transactions involving each pair of sender, subject and recipient as demonstrated in Table A.3 of Appendix A. For example, the transaction `Notice`:*s*–su is triggered by the sender to a subject, and results in transforming the memory of the subject.

5.5. *Process acknowledgments*

An acknowledgment provides the guarantee of information receipt. In a social setting, the mere act of a subject granting consent to disclose or the sender disclosing

the information may provide enough guarantee that the sender (respectively, recipient) is aware of the information. But this may not hold in a socio-technical setting. For example, the information may be held up in a message buffer not yet accessed by the reference or principal object. Hence, for the guarantee of delivery, an object may provide acknowledgment on receipt of the information. An acknowledgment can occur after the execution of `Request-`, `Sent-`, `Notice-` and `Consent`-related processes as shown in Table A.5 of Appendix A.

6. Disclosure Protocol Suite

The transactions in a disclosure protocol depend on its inherent processes:

- There are two variants of `Request`. These are `Request`, containing processes where the recipient requests for information from the sender, followed by the acknowledgment of request by the sender, and `Request2`, which only contains processes where the recipient requests for information from the sender, without any acknowledgment.
- There are eight variants of `Consent`. The first is `Consent1` where the sender seeks consent and then the subject grants consent. The second is `Consent2`, which involves a process where the subject grants consent without the sender initially seeking consent. The third variant is `Consent3` where the sender seeks consent, but is never granted by the subject. Furthermore, `Consent4` and `Consent5` extend `Consent1` and `Consent2`, respectively, with the acknowledgment of granted consent by the sender. Whereas, `Consent6`, `Consent7` and `Consent8` extend `Consent1`, `Consent3` and `Consent4`, respectively, with the acknowledgment of sought consent by the subject.
- Furthermore, there are two variants of `Sent`. First is `Sent1`, which contains processes where the sender discloses information to the recipient, and a receipt is then acknowledged by the recipient. The second is `Sent2`, and it only consists of a process where the sender discloses information to the recipient, without any acknowledgment of receipt.
- Finally, there are two variants for each pair of `Notice` transactions. For example, `Notice:`*s*–su1 contains processes where a sender notifies the subject of its (un)awareness and is thereafter acknowledged by the subject. Whereas, `Notice:`*s*–su2 only consists of a process where the sender notifies the subject of its (un) awareness, without any acknowledgment.

Altering the manner in which these different transactions are combined leads to disclosure protocols that uniquely transform the memory of an object. Ensuring that the memory of objects remains consistent after the execution of a disclosure protocol requires the assurance that only legitimate sequence of transactions is allowed in a disclosure protocol. Furthermore, each transaction may transform instances of an (un)awareness class differently. Hence, when two transactions are executed in

sequence, then a set of rules is necessary to determine how transformations resulting from one transaction override another.

6.1. *Memory consistency*

The legitimate sequence of transactions to ensure the memory of objects remains sound in a valid state is specified using the precedence (adjacency) matrix in Fig. 7. A `Request` transaction can only occur before `Sent` since the recipient can only

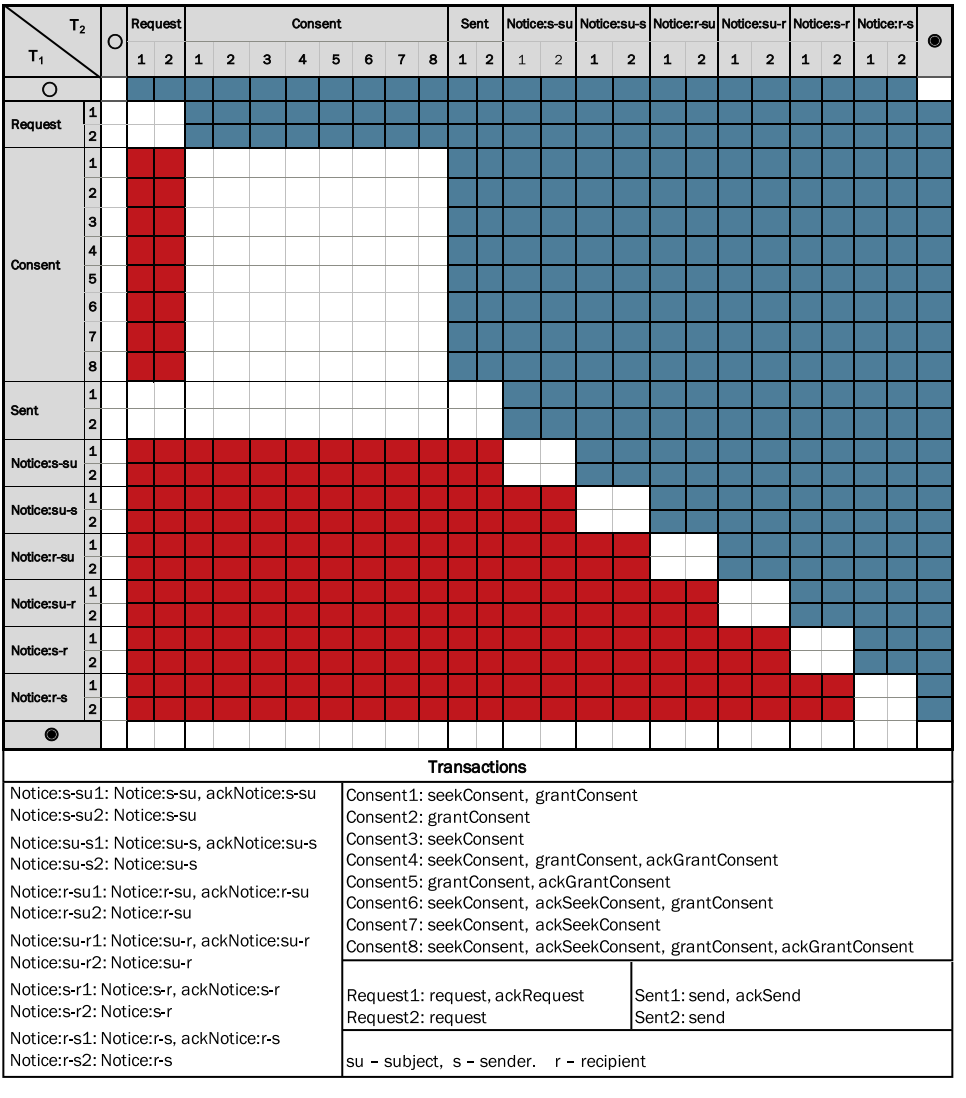


Fig. 7. Precedence (adjacency) matrix for a sequence of transactions. Shaded cells indicate that the transaction T_1 can be executed before T_2 in a trace.

request for information it is unaware. In contrast, a `Request` can occur before or after any variant of `Consent` or `Notice`. A disclosure protocol cannot contain more than one variant of `Consent`, and can only occur before `Sent`. This is important since seeking and/or granting consent when the information is already disclosed invalidates the purpose of `Consent`. Similarly, a protocol cannot contain more than one variant of `Sent`, which cannot occur before a `Request` or `Consent`. Finally, any variant of `Notice` can occur before or after any other transaction in a disclosure protocol. The resulting matrix is a state space of disclosure protocols, with traces containing a minimum of one and a maximum of nine transactions, respectively.

When two transaction processes are executed in sequence and the resulting transformation, for instance, of the same class and reference objects differs, then the process transformation that contains fewer negations and achieves lower awareness differential overrides the other. The general rules applied to determine how process transformations override each other are described in Table A.6 of Appendix A. When any variant of `Notice` occurs before a process, then the process overrides `Notice`. Conversely, when a process occurs before any variant of `Notice`, then the process is overridden by `Notice`. Interacting objects can also end up with different awareness differentials depending on the disclosure protocol. This difference is indicative of the varying levels of awareness that transactions in the protocol generate in each object. Based on this view, the memories of parties in an information flow are only consistent with each other when they are all fully aware. Whereas, partial awareness may introduce some level of inconsistencies based on varying awareness differentials and accounts for the uncertainty that objects may have about disclosed information.

6.2. *Properties of disclosure protocols*

The objective of privacy-preserving information disclosure is to transform the memory of parties involved to varying levels of awareness, ranging from being fully unaware to fully aware. Assume a simple information flow characterized by a sender (`Bao`) and a recipient (`Oz`), where `Bao` generates and discloses information about a subject (`Gor`). Also, consider that the recipient is revealed at the point consent is being sought and/or granted. Then each disclosure protocol trace may enable each party to achieve more or less awareness of the information. For example, the trace $\alpha = ($`Consent1`, `Sent1`, `Notice`:r–su, `Notice`:s–su, `Notice`:s–$r)$ would enable `Gor` to achieve more awareness compared to `Bao` and `Oz`. Whereas, the trace $\alpha = ($`Consent2`, `Notice`:su–r, `Request`, `Sent1`, `Notice`:r–$s)$ enables `Oz` and `Bao` to achieve more awareness compared to `Gor`. We leverage on three metrics to characterize the impact of disclosure protocols on the memory of objects.

6.2.1. *Unawareness level*

This is a measure of the extent an object remains unaware of information after disclosure. This metric is determined by comparing the awareness differential in the

memory of the object at t_0 (i.e. full unawareness) with the differential at t_n (after disclosure) as follows:

$$\texttt{UL}(p) = \frac{\left(\sum_{\mathbf{A}_i} \texttt{ADiff}(p, \mathbf{A}_i)\right)^{t_n}}{\left(\sum_{\mathbf{A}_i} \texttt{ADiff}(p, \mathbf{A}_i)\right)^{t_0}},$$

where $\mathbf{A}_i \in \{\mathbf{A1}, \mathbf{A2}, \mathbf{A3}, \mathbf{A4}\}$.

As awareness differential in the memory of an object at t_n turns towards zero, the unawareness level of the object also turns towards zero. Hence the information is more visible to the user. Otherwise, the object is less able to determine its awareness of disclosed information and/or the awareness of other objects about the disclosed information.

6.2.2. *The cost of a disclosure protocol*

This is a measure of the frequency at which instances in the memory of an object are switched from being untenable to tenable or vice versa by a disclosure protocol relative to another. Given α_i, the cost for an object is determined by

$$\texttt{Cost}(p, \alpha_i) = \left[\left(\frac{|\triangle M_p|_{\alpha_i} \times \texttt{HSize}_p}{|\triangle M_p|_{\alpha_{\max}} \times \texttt{HSize}_{\max}}\right)^{t_n}\right]_{\texttt{actual}} - \texttt{discount},$$

where $\alpha_{\max}$ is the disclosure protocol trace that generates the maximum number of memory transformations at t_n. Likewise, $\texttt{HSize}_p$ is the number of entries in p's interaction history. This represents the impact of p's evolving roles as the subject, sender and/or recipient of information up to t_n using α_i. Whereas, $\texttt{HSize}_{\max}$ is the maximum number of entries that can exist in the interaction history of an object at t_n given α_i. This cost is an indication of the actual overhead associated with using a disclosure protocol in designing interaction between objects in software. From an implementation viewpoint, the cost of a disclosure protocol indicates the resources and effort required to realize its design. Whereas, from an end-user viewpoint, more costly disclosure protocols may also be more disruptive. Finally, the actual cost can be forfeited with a discount factor that ranges between zero and the actual cost, which indicates the trade-off for increased or reduced unawareness levels.

The utility derived when the software is designed to enable interaction between objects based on a disclosure protocol depends on the privacy objective. When this objective is to increase awareness and the visibility of disclosed information at low cost, then privacy utility is computed as

$$\texttt{Util}_{\texttt{vis}}(p, \alpha_i) = [1 - \texttt{UL}(p)] - \texttt{Cost}(p, \alpha_i).$$

Alternatively, the objective is to increase unawareness and the secrecy of disclosed information at low cost. This is computed as

$$\texttt{Util}_{\texttt{sec}}(p, \alpha_i) = (\texttt{UL}(p)) - \texttt{Cost}(p, \alpha_i).$$

6.2.3. *Degree of freedom*

Constraining the disclosure behavior of users to a single protocol may inhibit the usability of the software. A design which implements a functional requirement using only one protocol provides minimum flexibility since users can only interact in one way. Likewise, the software design becomes more flexible if a functional requirement can be achieved using a set of alternative disclosure protocols that provide the same or acceptable range of privacy satisfaction. For example, the two traces $\alpha_a =$ (`Consent2`, `Sent1`, `Notice`:su–r) and $\alpha_b =$ (`Consent1`, `Sent1`, `Notice`:r–su) will generate similar unawareness levels and costs at $\texttt{UL}(\texttt{Gor}) = 0.08$ with $\texttt{Cost}(\texttt{Gor}, \alpha_a) = 0.69$ and $\texttt{Cost}(\texttt{Gor}, \alpha_b) = 0.69$, respectively. Hence, α_a and α_b provide the same level of privacy to `Gor` at the same cost. The number of disclosure protocols that can be used in software design to achieve an object's privacy objective is referred to as the degree of freedom (`DoF`) and determined by

$$\texttt{DoF}(p) = \frac{|[\texttt{Util}_{\texttt{obj}}(p, \alpha_i) \geq \texttt{Util}_{\texttt{lim}}]|}{|\texttt{DP}_{\texttt{matrix}}|},$$

where $\texttt{DP}_{\texttt{matrix}}$ is the set of disclosure protocols from the precedence matrix in Fig. 7. The function $\texttt{Util}_{\texttt{obj}}$ is the privacy utility realized by $\alpha_i \in \texttt{DP}_{\texttt{matrix}}$, when the objective is to enhance visibility or secrecy of disclosed information in the memory of p. Whereas, $\texttt{min} \leq \texttt{Util}_{\texttt{lim}} \leq \texttt{max}$ is the threshold on acceptable range of privacy utility. When the threshold is infinite, then every disclosure protocol trace in $\texttt{DP}_{\texttt{matrix}}$ can be used in enabling object interaction in the software design. In this case, $|\texttt{DP}_{\texttt{matrix}}| = 36,913,048$ with $\texttt{DoF}(p) = 1$. This is a relatively large state space and suggests the plethora of design options available to implement interaction between objects. But this state space can be pruned based on search constraints; for instance, by limiting the maximum and/or minimum number of transactions in a disclosure protocol, and also protocols that contain, exclude, start with and/or end with a specific transaction. For example, if the designer is only interested in disclosure protocols with a maximum of four transactions that contain any variant of `Sent` and a `Consent1`, then the awareness system will generate a reduced state space with $|\texttt{DP}_{\texttt{matrix}}| = 1580$.

7. Case Studies

Overall, designing software that preserves privacy is a balance between the information-flow settings and the disclosure protocol(s) used to enable interaction in the design. These settings are defined by properties such as the roles of objects associated with each flow along a path, and also the duality/uniqueness of such roles; whether the subject is the source of information being disclosed or the information is generated by or in custody of the sender; and finally, whether the subject and/or sender has identified the information recipients at the point consent is sought and/or granted. Whereas, the degree of freedom broadly indicates flexibility in the manner

software can be designed. In this section, we present two case studies to investigate the impact of these factors on real-world system implementation. The first study is a reverse engineering of the Twitter followership network to understand highlighted factors in its design. The second study is a scenario analysis involving a family of design patterns for realizing service-oriented software systems.

7.1. *Methodology*

Given a design that highlights the expected behavior of a system, we map observable interaction patterns noted in the software design to disclose protocol traces. We achieve this by the systematic analysis of functional specification as follows:

(1) Identify objects and their actions from behavioral specifications.
(2) Abstract roles of objects from identified interactions.
(3) Map actions to transactions to identify the associated disclosure protocol.

After the information-flow settings and disclosure protocols are discovered from a software design, the effectiveness of the design can be investigated using PriSAT. A designer can specify the disclosure protocols and information-flow settings over an interaction network. PriSAT then determines the extent a privacy objective is satisfied. Alternatively, given an information-flow setting, PriSAT determines the appropriate set of disclosure protocols that can be used to realize a privacy objective. We applied this methodology to evaluate the privacy-preserving capabilities in the design of private interactions on Twitter [51] and service-based software.

7.2. *The design of private interactions on Twitter*

Twitter is a social networking platform where users interact predominately via a followership network [51]. Users are expected to register with the service before they can interact with other users. Once registered, the interaction is fostered by a user following another user to gain visibility of the messages they tweet, retweet, like or reply. The relationship between a followed user and the follower is not symmetric. Also, disclosure behavior is dependent on whether the users choose to interact privately or publicly in their privacy configurations. In this study, we focus on a scenario where a user interacts with its followers privately as shown in Fig. 8.

When a user follows another user, then the follower is the recipient and the followed user is the sender. This action implies that the recipient is requesting for the visibility of all actions executed by the sender on a message. When the followed user account is set as private, then a followership request from the follower has to be explicitly confirmed by the followed user. This corresponds to `Request1` transaction considering that a followership request from a recipient matches the `request` process, while a confirmation from the sender is `ackRequest`. When a message is tweeted, the follower is the recipient while the followed user is the source of the tweet and therefore the sender and subject. The follower may like a private tweet, while a

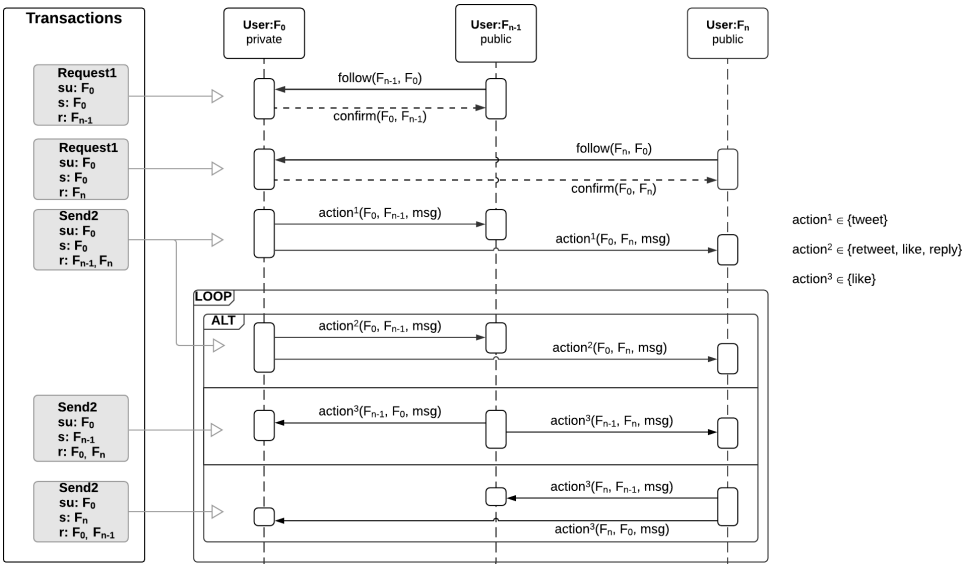


Fig. 8. Sequence diagram of the followership-based interactions on Twitter platform and the associated transactions.

reply is considered a new message and not bound by the privacy settings of the followed user. A "like" action can be initiated by the followed user or follower, who, respectively, assumes the role of a sender. If initiated by a followed user, then its followers are the recipients. Whereas, the visibility of an action on a tweet that is initiated by a follower is dependent on whether or not the tweet is private. For a private tweet, the followed user and its other confirmed followers each assume the role of a recipient. These actions correspond to a `Sent2` transaction, since the propagation of a message by a sender is not acknowledged by the respective recipients. Hence, the corresponding disclosure protocol for interacting privately on Twitter is $\alpha_1 = (\texttt{Request1}, \texttt{Sent2})$, which matches the scenario where a private user is followed by other users, then the followed user or follower tweets, likes, replies or retweets a message.

7.2.1. *Followership design on Twitter*

There are diverse followership scenarios in a private setting. A user may choose to follow or unfollow another user at any time and a message tweet does not always attract the same amount of likes from a set of followers. This results in a changing followership network for every message that is tweeted. The scenario in Fig. 9 represents a user F_0 that tweets a private message to five of its followers F_1–F_5. This results in five information-flow paths with an interaction history where F_0 assumes a dual role of the subject and sender five times, with each follower being a recipient once. At this point, the followership design in Fig. 8 yields a maximum expected

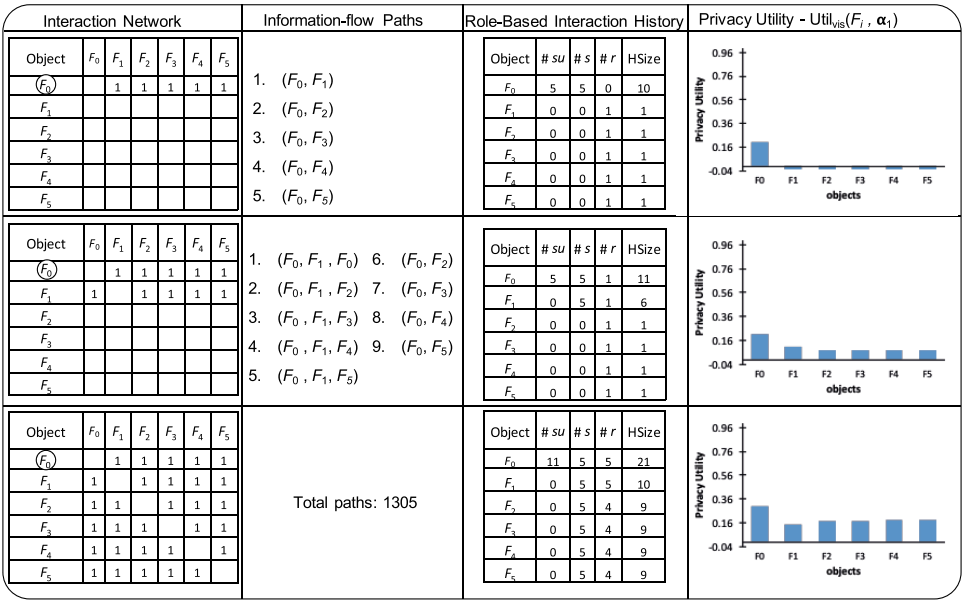


Fig. 9. Twitter interaction after a private message is tweeted from F_0 to its five followers and the subsequent like action executed by each follower using α_1 and $\texttt{discount} = 0$.

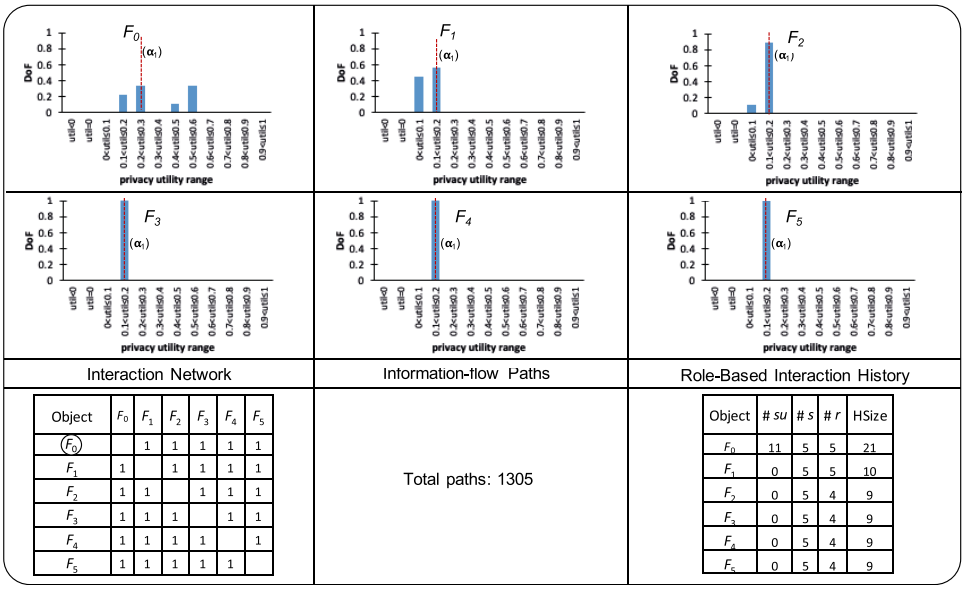
positive utility of 0.2 for F_0 when α_1 is used to enable interaction and the privacy objective to reduce unawareness levels. Whereas, the followers F_1–F_5 all derive negative utility with the mean of 0.01 across all users. When F_1 subsequently likes the tweet, the interaction network is extended to nine information-flow paths, with F_0 assuming an additional role of a recipient while F_1 assumes the role of a sender five times. This action improves the mean privacy utility to 0.1 for all users in the network. Overall, it is observed that the unawareness level of users reduces and privacy utility increases with each like action on a tweet. A maximum mean utility of 0.2 is reached when all the followers have liked the tweet. At this point, a total of 1305 paths would have been generated.

Based on the outlined scenario, the research question is whether there is an alternative followership design that improves the expected utility for users given a privacy objective. We note that an objective to maximize information visibility cannot necessarily be realized by using a trace with high number of transactions and processes. For example, the trace $\alpha_{\texttt{max}} = (\texttt{Request1}, \texttt{Consent8}, \texttt{Sent1}, \texttt{Notice}{:}s\text{–su1}, \texttt{Notice}{:}\text{su}\text{–}s1, \texttt{Notice}{:}s\text{–}r1, \texttt{Notice}{:}r\text{–}s1, \texttt{Notice}{:}r\text{–su1}, \texttt{Notice}{:}\text{su}\text{–}r1)$ results in $\texttt{UL}(F_0) = 0$, $\texttt{Cost}(F_0, \alpha_{\texttt{max}}) = 1$ and $\texttt{Util}_{\texttt{vis}}(F_0, \alpha_{\texttt{max}}) = -1$. This represents a state where F_0 achieves full awareness but at the maximum cost and hence the worst negative utility. Utilizing $\alpha_{\texttt{max}}$ for interaction is only viable when F_0 discounts its associated cost of interaction. Likewise, once the information is disclosed, it is impossible for a privacy objective of maximizing information secrecy to be realized with

a utility of 1. This is because any disclosure protocol used will result in some unawareness reduction in either the subject, sender and/or recipient. Thus, the impact of these contending factors can be mitigated by leveraging on the utility values to determine the extent a privacy objective is satisfied.

To investigate an alternative design, DoF analysis was carried out on the outlined scenario using PriSAT. The analysis focused on identifying potential lightweight refactorings where inherent transactions in the design are preserved but augmented with variants of `Consent` or `Notice`. In this way, the functional properties of the followership network remain unchanged. PriSAT was used to search the precedence matrix for traces that start with `Request1`, may contain any variant of `Consent` and end with `Sent2`. The outcome was a reduced state space with $|\texttt{DP}_{\texttt{matrix}}| = 9$ which includes α_1 and the traces (`Request1`, `Consent`$[1|2|\ldots|8]$, `Sent2`). Also, since the followership network is bound by a confirmation of follow request by the followed, it was assumed that the subject and sender have identified the information recipient at the point the consent is sought.

Figure 10 illustrates the outcome of DoF analysis based on traces that matched our `Consent` search criteria. For F_0, the traces (`Request1`, `Consent`$[2|3|5|7]$, `Sent2`) generated a utility greater than α_1 which ranged between 0.4 and 0.6. Whereas, (`Request1`, `Consent`$[1|6]$, `Sent2`) offered utilities similar to α_1 while (`Request1`, `Consent`$[4|8]$, `Sent2`) did not offer better utilities compared to α_1. For F_1, the traces (`Request1`, `Consent`$[1|4|6|8]$, `Sent2`) generated lesser utility values ranging between



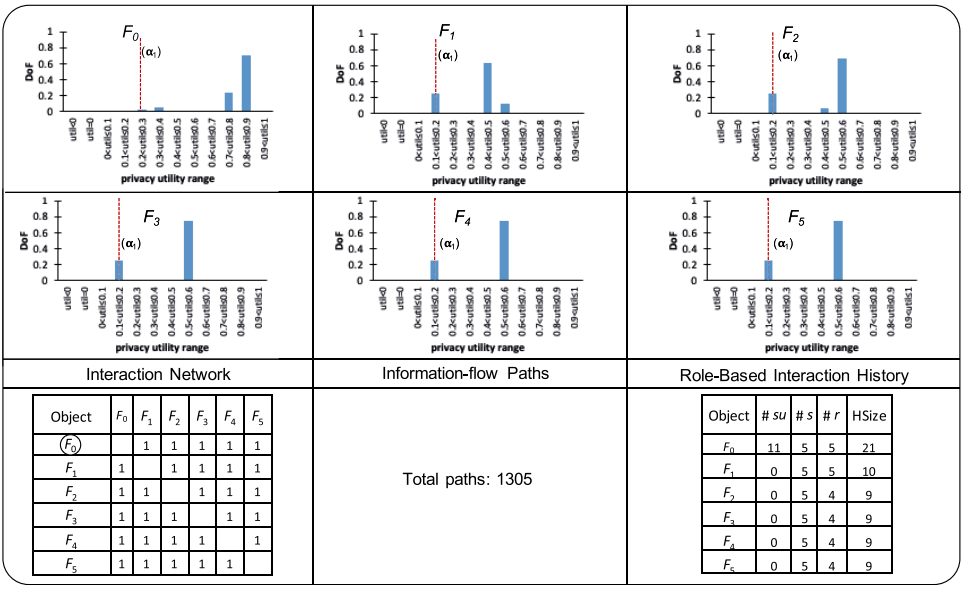
Object	F_0	F_1	F_2	F_3	F_4	F_5
F_0		1	1	1	1	1
F_1	1		1	1	1	1
F_2	1	1		1	1	1
F_3	1	1	1		1	1
F_4	1	1	1	1		1
F_5	1	1	1	1	1	

Object	# *su*	# *s*	# *r*	HSize
F_0	11	5	5	21
F_1	0	5	5	10
F_2	0	5	4	9
F_3	0	5	4	9
F_4	0	5	4	9
F_5	0	5	4	9

Fig. 10. The DoFs in realizing categories of privacy utilities for a private message tweet from F_0 to F_1–F_5 with every follower responding with a like action. The dotted marker indicates the DoF classification for α_1 and the $\texttt{DP}_{\texttt{matrix}}$ contains the traces where α_1 is augmented with a variant of `Consent` transaction.

0 and 0.1, while (`Request1`, `Consent`[2|3|5|7], `Sent2`) yields the same utility as α_1. Likewise for F_2–F_5, the traces that contained variants of `Consent` resulted in the same utility as α_1. A similar pattern of utility variance with DoF was observed irrespective of the number of followers associated with F_0. It is therefore concluded that augmenting Twitter fellowship design with variants of `Consent` transactions only enhances privacy utility for the subject that tweets a private message. Whereas, the utility for its followers is not improved.

The second refactoring involved augmenting existing design with a combination of acknowledged `Notice` transactions. Hence, PriSAT was used to search the precedence matrix for traces that start with `Request1`, followed by `Sent2` and may end with a combination of one or more forms of `Notice` transactions without acknowledgment. This generated a state space with $|\mathtt{DP}_{\mathtt{matrix}}| = 64$ which contained α_1 and the traces $(\mathtt{Request1}, \mathtt{Sent2}, 2^{\mathtt{X}})$, where $\mathtt{X} = \{$`Notice`:s–su2, `Notice`:su–s2, `Notice`:r–su2, `Notice`:su–r2, `Notice`:s–r2, `Notice`:r–s2$\}$. The performance of α_1 against `Notice`-related traces is shown in Fig. 11. For F_0, all traces generated utility values greater than α_1. Furthermore, three traces consisting of $(\mathtt{Request1}, \mathtt{Sent2}, 2^{\mathtt{X}^+})$, where $X^+ = \{$`Notice`:su–r2, `Notice`:su–s2$\}$ generated utility values ranging between 0.3 and 0.4, respectively. The remaining 60 traces provided significantly improved utility values that ranged between 0.7 and 0.9, respectively. Likewise, for F_1–F_5, 15 traces consisting of $(\mathtt{Request1}, \mathtt{Sent2}, 2^{\mathtt{X}^{++}})$, where $\mathtt{X}^{++} = \{$`Notice`:s–su2, `Notice`:su–s2, `Notice`:r–su2, `Notice`:su–r2$\}$ had no improved performance over α_1.



Object	F_0	F_1	F_2	F_3	F_4	F_5
F_0		1	1	1	1	1
F_1	1		1	1	1	1
F_2	1	1		1	1	1
F_3	1	1	1		1	1
F_4	1	1	1	1		1
F_5	1	1	1	1	1	

Object	# *su*	# *s*	# *r*	HSize
F_0	11	5	5	21
F_1	0	5	5	10
F_2	0	5	4	9
F_3	0	5	4	9
F_4	0	5	4	9
F_5	0	5	4	9

Fig. 11. The DoFs in realizing categories of privacy utilities for a private message tweet from F_0 to F_1–F_5 with every follower responding with a like action. The dotted marker indicates the DoF classification for α_1 and the $\mathtt{DP}_{\mathtt{matrix}}$ contains traces where α_1 is augmented with combinations of `Notice` transactions.

Whereas, the remaining 48 traces provided better utility values that ranged between 0.4 and 0.6, respectively. Again, a similar pattern of utility variance with DoF was observed irrespective of the number of followers associated with F_0. Thus, given an objective to minimize user's unawareness, privacy utility on Twitter followership design for exchanging private tweets can be enhanced by augmenting inherent disclosure protocol with $\mathtt{DP}_{\mathtt{twitter}} = (\mathtt{Request1}, \mathtt{Sent2}, \mathtt{DP}_{\mathtt{au}})$, where $\mathtt{DP}_{\mathtt{au}}$ represents the combination of `Notice` transactions from the set $\mathtt{DP}_{\mathtt{au}} = \mathtt{X} - \mathtt{X}^{++}$ and $\mathtt{X}^{+} \subseteq \mathtt{X}^{++}$.

7.2.2. *Discussion*

A key insight is that the maximum privacy utility inherent in Twitter followership design is marginal compared to an alternative design that is augmented with a subset from a combination of `Notice` transactions. This makes the latter a preferred design when the objective is to broadly maximize the visibility of information in the network. For example, the relative improvements in privacy utilities for F_0–F_5 can be observed in Fig. 12 where $\alpha_1' = (\mathtt{Request1}, \mathtt{Sent2}, \mathtt{Notice}{:}s\text{–}r2) \in \mathtt{DP}_{\mathtt{twitter}}$ is used for interaction, compared to α_1 in Fig. 9. A maximum mean utility of 0.6 is reached when all the followers have liked a tweet using α_1' compared to 0.2 that is derived using α_1. We note that refactoring an existing design to achieve a privacy objective may further require domain-specific design choices. For instance, refactoring the followership design in Fig. 8 to realize α_1' will require that a sender does not only disclose a message to the recipient, but also inform the recipient of other recipients to which it discloses the same message (see the design extension in Fig. 13).

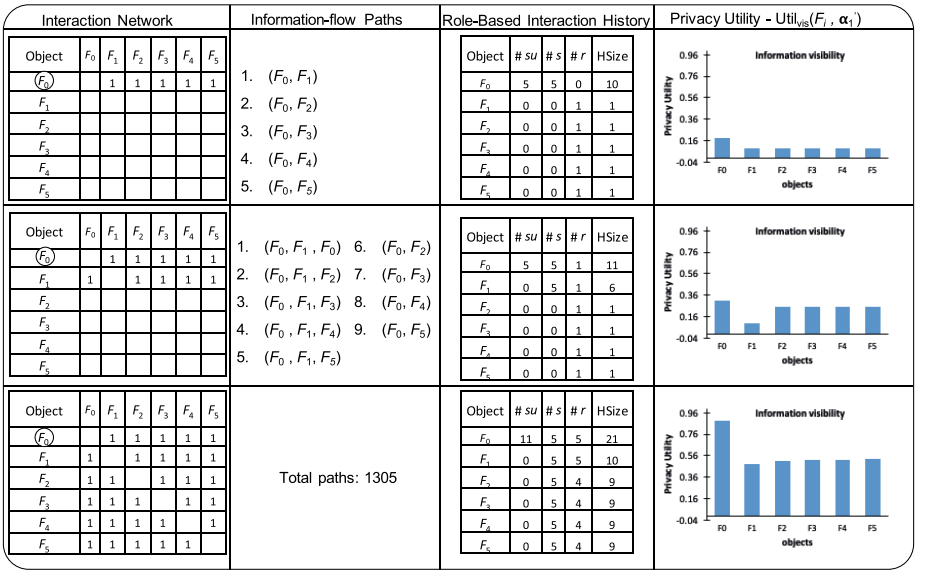


Fig. 12. Twitter interaction after a private message is tweeted from F_0 to its five followers and the subsequent like action executed by each follower using $\alpha_1' = (\mathtt{Request1}, \mathtt{Sent2}, \mathtt{Notice}{:}s\text{–}r2)$.

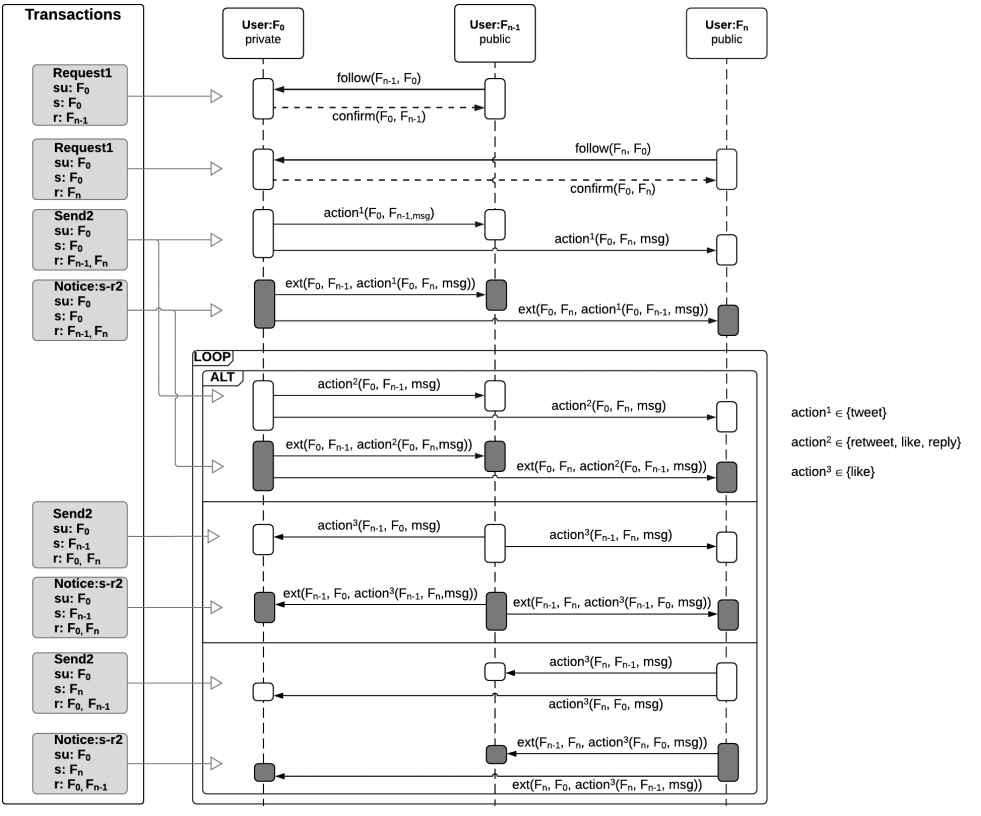


Fig. 13. Refactoring Twitter followership design to realize `Notice`:s–$r2$.

Alternatively, a privacy objective may be to reduce visibility, for example, making information less visible for all users in the network or a subset of users. For the former, α_1 is the preferred disclosure protocol compared to α_1'. Both protocols yield mean positive and negative utilities of 0.35 and -0.07, respectively, when the privacy objective is to maximize information secrecy. Whereas, when the privacy objective is to achieve varying information visibilities, then a followership design which uses a single disclosure protocol to foster interaction does not necessarily provide an equal amount of privacy utility for each user on the network. This is illustrated in Figs. 9 and 12 where the privacy utility for the subject F_0 tends to differ significantly from other users.

These findings suggest privacy in Twitter's software design can be enhanced in one of the two ways. The first is informing users of their changing privacy utility as their personal information flows from one user to another in the network. Users can then adjust their disclosure behavior to mitigate emerging privacy concerns. Second, variability in privacy utility can be managed by enabling users to specify their privacy objectives and expected utilities. Disclosure protocols are then dynamically selected by the platform during interaction to satisfy a privacy objective.

We have only considered lightweight refactorings where inherent transactions in the design are preserved but augmented with variants of `Consent` or `Notice` transactions. Whereas, there are other refactoring options; for example, considering a combination of `Notice` transactions before and after `Sent1` is executed, as well as traces that contain `Consent` and `Notice` with varying precedence. Again, we assumed that a message has only one subject and after a tweet, the authorship and attribution of the message do not change. Whereas, it is possible to have a message associated with multiple subjects; for example, when the message mentions another user via a UserTag. The structural/semantic changes to the message make the user that is tagged a co-owner and also a subject.

7.3. *The privacy analysis of service-based software design*

Interactions between components in distributed software applications are often orchestrated as services. A service is a discoverable software entity that can exist as a single instance and interacts synchronously or asynchronously with applications and other services through a loosely coupled communication model [42]. This concept is based on a software architectural style that defines an interaction between three primary entities: the service producer, who publishes a service description and provides the implementation for the service; a service consumer, who uses the service; and the service broker that enables interaction between the producer and consumer [30]. There are two main interaction patterns involving identified entities [19, 50]. The first is a message pattern where all communication (information flow) that occurs between a service producer and a consumer is mediated by a service broker [42]. The alternative pattern is where information flow occurs in a peer-to-peer fashion following the "register–find–bind–execute" paradigm [36]. The producer registers a service contract in a public registry that exists on the broker. This registry is queried by consumers to find services that match certain criteria. If the registry has such a service, the broker provides the consumer with the contract and an endpoint address to bind directly with the producer. Both interaction patterns are often constrained by Quality-of-Service (QoS) assurances to satisfy certain nonfunctional requirements. More importantly, these patterns offer different interaction patterns amongst service objects. In this subsection, we articulate a subset of these scenarios to gain insights into their privacy-preservation capabilities.

7.3.1. *Broker-mediated interaction pattern*

In broker-mediated service interaction, information flow is initiated via the broker using a push/pull mechanism or a hybrid of both as shown in Fig. 14. When the producer pushes a message to the broker, then the producer grants consent for the broker to disclose the message to any set of consumers. Where the QoS necessitates that the broker responds with acknowledgment after the consent is granted, then the matching transaction it is `Consent2`, otherwise it is `Consent5`. Alternatively, the

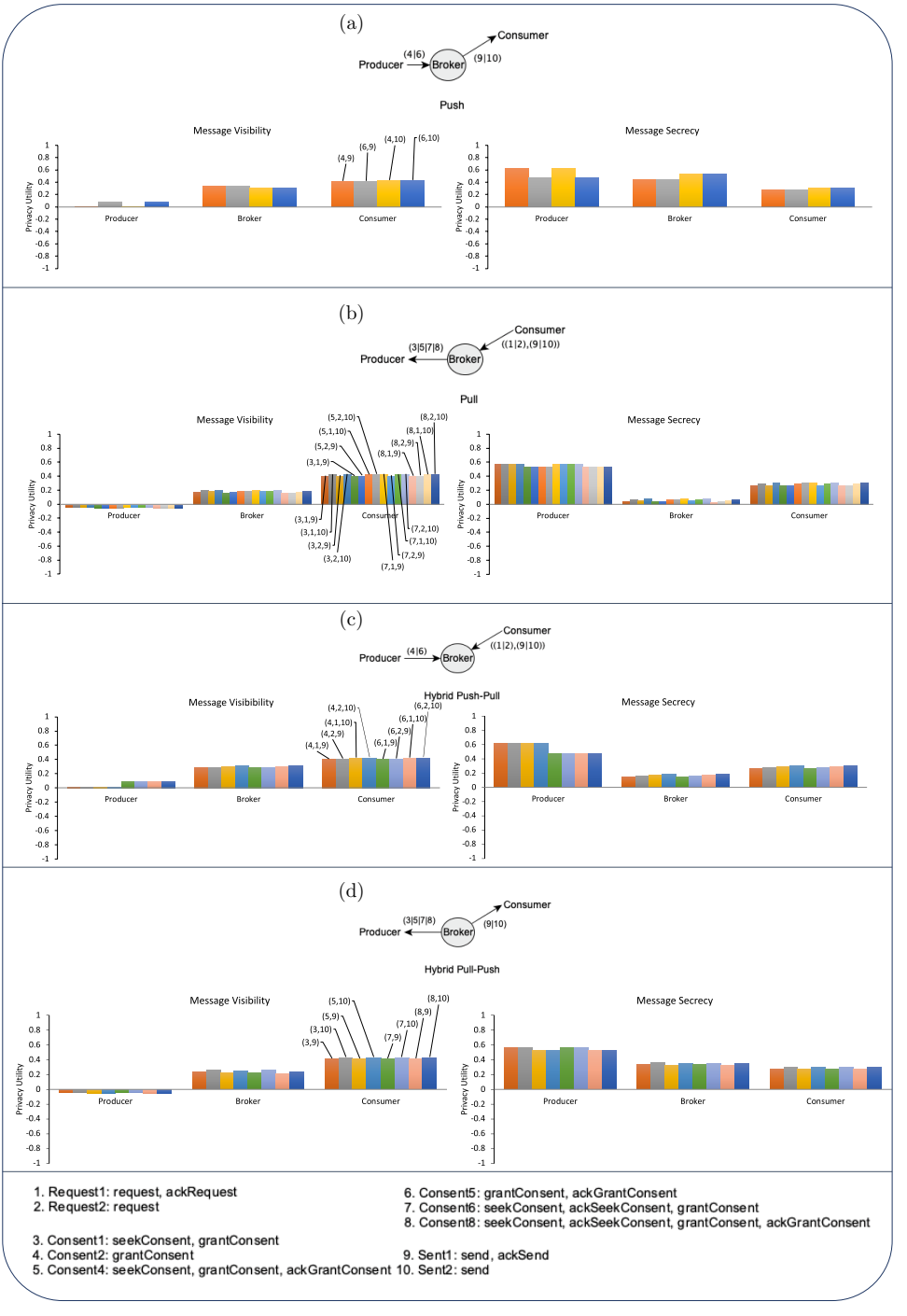


Fig. 14. Analysis of broker-mediated architectural models for service-based interaction.

broker may pull the message from the producer. In this case, the broker is seeking consent from the producer to disclose the message to any set of consumers, with consent subsequently granted by the producer. Typically, the broker and producer have no knowledge of the consumer at this point. The matching transaction is either `Consent1`, `Consent4`, `Consent6` or `Consent8`, and it depends on whether or not the QoS guarantees acknowledgment. The broker may then send the message to the consumer via a push mechanism. The matching transaction is either `Sent1` or `Sent2` and again depends on the associated QoS assurances. Alternatively, the consumer may pull the message from the broker. In this case, the consumer first requests for a message about the producer from the broker. This matches a variant of `Request` transaction. Subsequently, the broker sends the requested message to the consumer to match `Sent1` or `Sent2` transaction depending on QoS assurances. MQTT, popularly used to implement IoT device-to-device interaction, is an example of a standard messaging specification based on broker-mediated interaction model [18]. Other examples include the Java Message Service (JMS) [33] and its implementation such as Apache ActiveMQ [52]. There are a number of interaction scenarios based on this pattern. These are as follows.

Scenario 1. Interaction between parties is achieved strictly via a push mechanism. The producer grants message disclosure consent to the broker. Subsequently, the message is disclosed by the broker to the consumer. As illustrated in Fig. 14(a), interaction between parties is realized using one of the four disclosure protocols from (`Consent`[2|5], `Sent`[1|2]).

Scenario 2. Interaction between parties is achieved strictly via a pull mechanism. The broker first seeks consent to disclose a message from the producer. Subsequently, consent is granted by the producer. Afterwards, the consumer requests the message from the broker. Finally, the message is disclosed by the broker to the consumer. Interaction is realized using one of the 16 disclosure protocols from (`Consent`[1|4|6|8], `Request`[1|2], `Sent`[1|2]) as illustrated in Fig. 14(b).

Scenario 3. Interaction between parties is achieved via a hybrid push–pull mechanism. First, the producer grants message disclosure consent to the broker. Afterwards, the consumer requests for the message from the broker. Finally, the message is disclosed by the broker to the consumer. For this case, the interaction is realized using one of the eight disclosure protocols from (`Consent`[2|5], `Request`[1|2], `Sent`[1|2]) as illustrated in Fig. 14(c).

Scenario 4. Interaction between parties is achieved via a hybrid pull–push mechanism. The broker first seeks consent to disclose a message from the producer. Subsequently, consent is granted by the producer. Finally, the message is disclosed by the broker to the consumer. As illustrated in Fig. 14(d), the interaction between parties is realized using one of the eight disclosure protocols from (`Consent`[1|4|6|8], `Sent`[1|2]).

7.3.2. *Broker-facilitated interaction pattern*

When interaction occurs in a peer-to-peer fashion, then message passing only takes place during binding and execution. The broker facilitates this by enabling the producer and consumer to discover each other via a push or pull mechanism, as well as the hybrid of both. This is initiated when the producer pushes a contract which typically contains the services it produces and its uniform resource identifier to the broker. This is synonymous to a producer notifying the broker of its capabilities. Depending on whether there exist QoS guarantees of acknowledgment, the matching disclosure transaction is a variant of `Notice`:su–*s* where the subject is a producer and the sender a broker. Alternatively, the broker can initiate a pull request for contracts from the producer, who then responds by publishing the contracts on the broker. In this case, the matching disclosure transaction is a variant of `Request`, which is then followed by a variant of `Notice`:su–*s*. Likewise, the consumer discovers service contracts by initiating a pull request on the broker's matching variants of `Request`, followed by a `Notice`:*s*–*r* where the broker acts as a sender and the consumer a recipient. The broker may also push service contracts to the consumer without prior request.

Once the consumer discovers a service contract, it then binds with the producer. At this point, the consumer invokes or initiates an interaction with the producer using the binding details in the service contract to locate, contact and invoke the service. The matching disclosure transaction is a variant of `Request`. The successful invocation of a service typically results in a functional execution by the producer and the result is returned to the consumer. This matches `Sent1` or `Sent2` transactions, depending on whether there exist QoS guarantees of acknowledgment. Examples of distributed software frameworks and standards based on broker-facilitated interaction patterns include the Simple Object Access Protocol (SOAP), Representational State Transfer (REST) for Web services and its reference implementations such as Java API for RESTful Web Services (JAX-RS) [31] and Jersey [32]. Interaction scenarios based on this service model include the following.

Scenario 5. Interaction between parties is achieved using a push with binding. The producer first notifies broker of a service contract. This is followed by the broker notifying a consumer of the producer's service contract. The consumer then makes a message request to the producer based on the service contract. Finally, the message is disclosed by the producer to the consumer. As illustrated in Fig. 15(a), interaction between parties is realized using one of the 16 disclosure protocols from (`Notice`:su-*s*[1|2], `Notice`:*s*–*r*[1|2],`Request`[1|2], `Sent`[1|2]).

Scenario 6. Interaction between parties is achieved using a pull with binding. The broker first requests for a contract from the producer, who responds by notifying the broker of a service contract. Next, the consumer requests for matching contract from the broker's registry, with the broker responding by notifying the consumer of the contract offered by the producer. The consumer then makes a message request to the

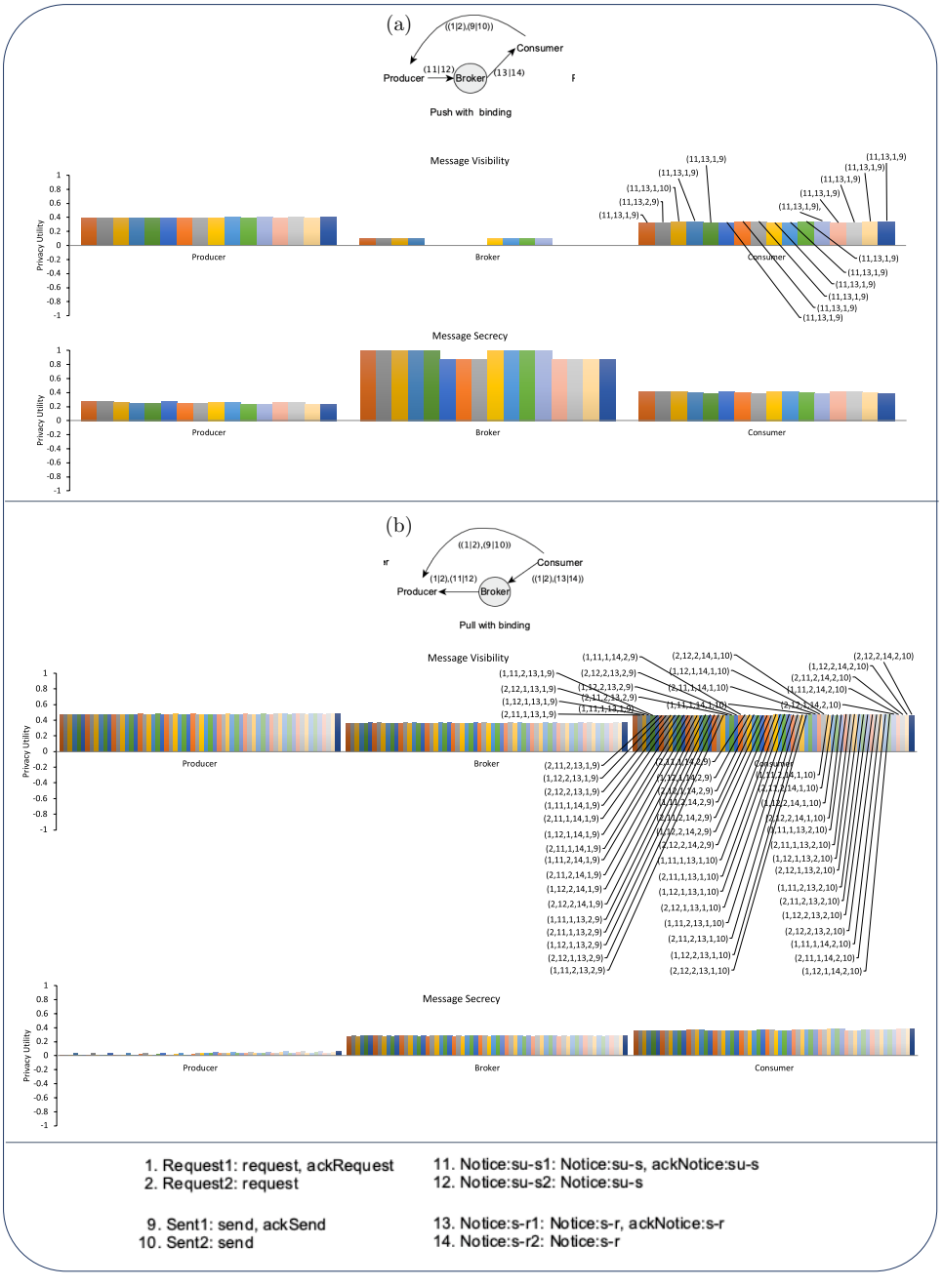


Fig. 15. Analysis of broker-facilitated architectural models for service-based interaction. Privacy utilities are determined without cost discount on the disclosure protocols.

producer based on the service contract. Finally, the message is disclosed by the producer to the consumer. This interaction scenario is realized using one of 64 disclosure protocols from (`Request`[1|2], `Notice`:su-s[1|2], `Request`[1|2], `Notice`:s–r[1|2], `Request`[1|2], `Sent`[1|2]) as illustrated in Fig. 15(b).

Scenario 7. Interaction between parties is achieved using a hybrid push–pull with binding. The producer first notifies broker of a service contract. Next, the consumer requests for a matching contract from the broker's registry, with the broker responding by notifying the consumer of the contract offered by the producer. The consumer then makes a message request to the producer based on the service contract. Finally, the message is disclosed by the producer to the consumer. This is illustrated in Fig. 16(a), and realized using one of the 32 disclosure protocols from (`Notice`:su-s[1|2], `Request`[1|2], `Notice`:s–r[1|2], `Request`[1|2], `Sent`[1|2]).

Scenario 8. Interaction between parties is achieved using a hybrid pull–push with binding. The broker first requests for a contract from the producer, who responds by notifying the broker of a service contract. The broker then notifies the consumer of the contract offered by the producer. The consumer then makes a message request to the producer based on the service contract. This ends with the message being disclosed by the producer to the consumer. This is illustrated in Fig. 16(b), and realized using one of the 32 disclosure protocols from (`Request`[1|2], `Notice`:su-s[1|2], `Notice`:s–r[1|2], `Request`[1|2], `Sent`[1|2]).

7.3.3. *Discussion*

A key insight from carrying out the scenario analysis of service interaction patterns relates to the DoF in realizing a design. Scenarios 1–4 can be realized using one of the 4, 16, 8 and 8 disclosure protocols, respectively. Whereas, Scenarios 5–8 can be realized using one of the 16, 64, 32 and 32 disclosure protocols, respectively. Hence, it can be inferred that the flexibility in designing a service-based distributed software is dependent on whether the interaction is modeled using push or pull mechanism or a combination of both, and also with or without message binding.

Furthermore, there is a limit to the privacy-preserving capability of each interaction pattern. It is observed that the choice to design a software system based on a pattern may inhibit or enhance the ability of producer, broker and/or consumer to realize a privacy objective. For example, the plot in Fig. 14(a) shows privacy utilities realized by the producer, broker and consumer for the visibility and secrecy objectives in Scenario 1. When the objective is to maximize message visibility, then the maximum privacy utility realized by the producer is 0.01, and is achieved using (`Consent5`, `Sent`[1|2]). This is insignificant, compared to the mean utilities of 0.33 and 0.43 realized by the broker and consumer across all the disclosure protocols that can be used in the scenario. Conversely, the same producer would realize a maximum privacy utility of 0.63 using (`Consent2`, `Sent`[1|2]) when the objective is to maximize secrecy. The broker and consumer also realize relatively significant privacy utilities.

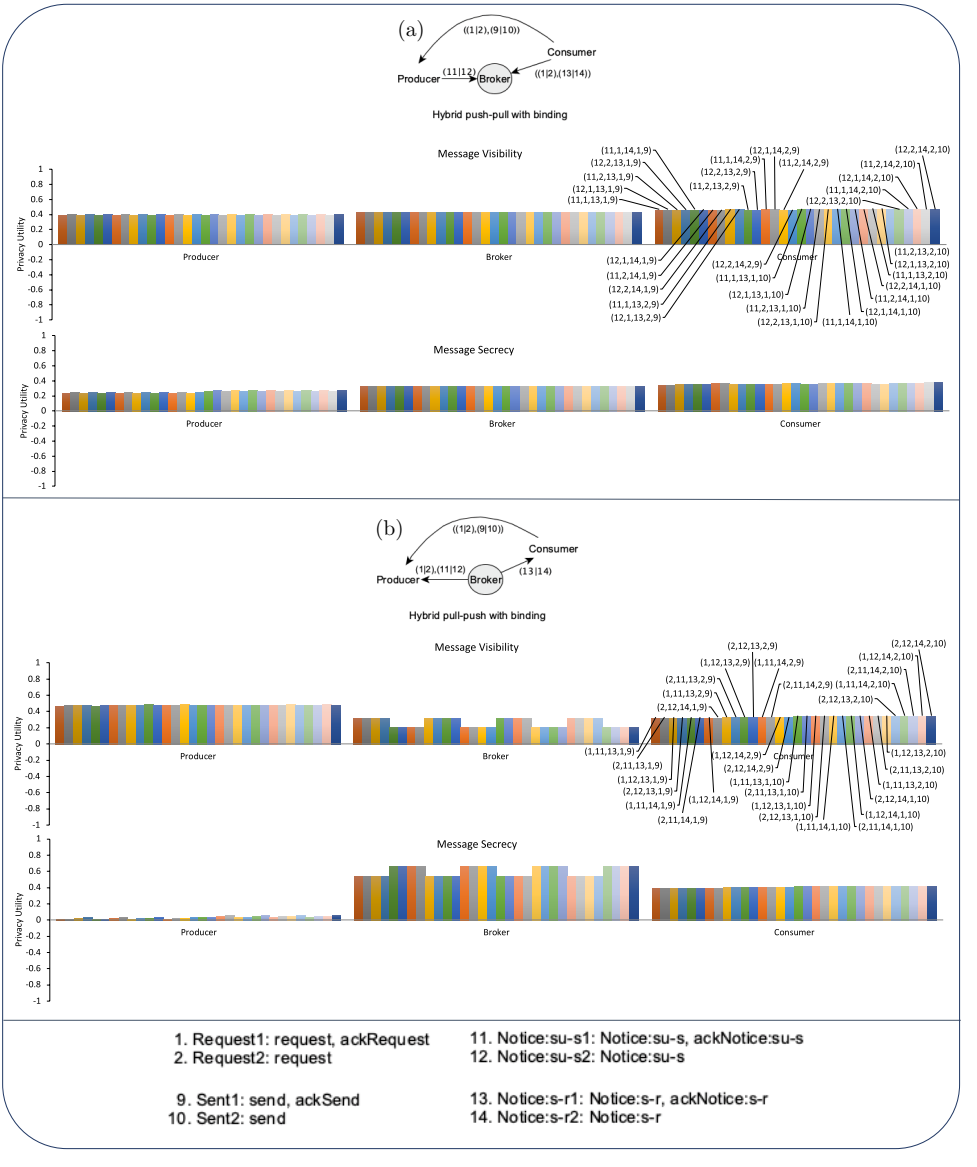


Fig. 16. Analysis of broker-facilitated architectural models for service-based interaction. Privacy utilities are determined without cost discount on the disclosure protocols.

Hence, it is concluded that when a service-oriented design is achieved strictly using a push mechanism, then the maximum privacy utility is realized when the objective is to maximize message secrecy across interacting parties.

Again, the plot in Fig. 14(b) illustrates the privacy utilities realized by parties in Scenario 2. It is observed that any disclosure protocol used in this scenario offers a

negative privacy utility to the producer when the objective is to maximize message visibility. Whereas, the broker and consumer will satisfy the same objective irrespective of the disclosure protocol used, with mean utilities of 0.19 and 0.42, respectively, across the 16 disclosure protocols that can be used in this scenario. This suggests that a pull mechanism is less suitable for a service-oriented design when the objective is to maximize the extent the information is visible to the producer. Conversely, when the objective is to maximize secrecy, mean privacy utilities of 0.56, 0.07 and 0.30 are realized by the producer, broker and consumer, respectively. This suggests that it is less efficient to achieve a service-oriented design using a pull mechanism when the privacy objective is to maximize the extent the message remains secret to the broker. Similar reasoning can be applied to Scenarios 3–8 to evaluate their suitability in realizing a privacy objective in a software design.

Observed variance in privacy utilities implies that instantiating a pattern in a service-oriented design may involve some compromise in privacy by either the producer, broker and/or consumer. A plot of mean utilities across all the disclosure protocols for each analyzed scenario is illustrated in Fig. 17. When the software is designed based on pull, push or a hybrid of both (marked S1–S4, respectively, in Fig. 17), then the broker and consumer would significantly know more about the message than the message producer. Similarly, when the design strategy is to ensure that the message being disclosed is least visible to the broker, then the appropriate design choice is a push with binding interaction pattern (marked S5 in Fig. 17). Alternatively, the design strategy may be to ensure that the message disclosed is equally visible to the parties involved. Then the appropriate design choice is a hybrid push–pull with binding interaction pattern (marked S7 in Fig. 17). This pattern offers the least variance in privacy utility between interacting parties.

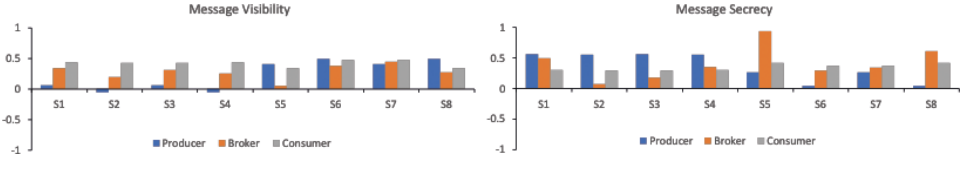


Fig. 17. Mean utilities across all the disclosure protocols for Scenarios 1–8.

This exploration of service-oriented design patterns provides insights on how a designer can select a design pattern based on a privacy objective, the privacy utility that a pattern provides and the satisfaction of desired functional requirements.

8. Threats to Validity and Future work

In our case studies, we leveraged on alternative documentation available in the public domain and observed the behavior of running systems to build the interaction

models. In future work, we intend to automate this task by augmenting the design artifacts familiar to designers in their daily work with insights on privacy implications. We expect that such automation will reduce the knowledge gap required to consider privacy during early-stage software design.

Our technique precludes factors such as the adversarial or cooperative tendencies of objects and also the level of sensitivity of disclosed information. Also, memory transformations during object interaction and implied awareness are solely determined by the executed disclosure protocols. This means that by only relying on transactions in the disclosure protocol suite, there are memory states that cannot be reached from an assumed initial state of full unawareness. For example, it is impossible to realize a disclosure protocol that renders all elements in the memory of a principal tenable. Whereas, it is easy to see that such a memory state is unintuitive from a socio-technical viewpoint, since this will infer an object denies its awareness after disclosure even though it is tenable that it is aware. Hence, it can be concluded that a disclosure protocol that makes an object to realize such a state should not be allowed in a software design. The open research question is to determine whether all unreachable memory states are also unintuitive from a socio-technical context, and therefore not relevant for privacy management.

The analysis of a disclosure protocols state space is a *reachability problem* of determining whether there is a disclosure protocol that makes a certain awareness state reachable from an initial state of full unawareness. Addressing this problem requires: (1) identifying all memory states that an object can assume based on Table 1; (2) determining which identified states are intuitive from a socio-technical viewpoint; (3) inferring whether there is a disclosure protocol that makes the state reachable; and finally (4) the impact that such reached/unreachable state has on privacy. A memory leak then exists when there is an unreachable state that is intuitive from a socio-technical viewpoint. Otherwise, the disclosure protocol suite can be considered as complete. Therefore, we do not claim in this research that the disclosure protocol suite is complete and prevents all memory leaks. Making such a claim by addressing highlighted tasks is beyond the scope of this paper and the focus of future work.

A broader picture of the relationship between end users, software designers and regulations is multi-dimensional, whereas this research only sets the foundation for understanding this relationship from a designer's viewpoint. Finally, although we assume that the (un)awareness modeled in objects is the same as their users', this deterministic assumption may sometimes not hold.

9. Conclusions

This paper presents a technique for analyzing the privacy-preserving capabilities of software design. First, possible world semantics are used to demonstrate how the memory of user objects in a behavioral design representation transforms during interaction, where an object's memory is defined in terms of what its respective users are aware or unaware of the disclosed information. The more unaware an object is

about the disclosed information, the more secret the information is to its user. Conversely, the more aware an object is about the disclosed information, the more visible the information is to its user. Privacy engineering during software design then involves determining the appropriate disclosure protocol that can be used in the design to ensure a level of information secrecy or visibility when objects are interacting. Hence, we define a disclosure protocol that constitutes information `Request`, `Consent`, `Sent` and `Notice` transactions, and characterize ensuing object memory transformations when any of the transaction is triggered as part of information disclosure. Finally, given a privacy objective to maximize information visibility or secrecy, a privacy awareness system is used to determine the privacy utility that users derive when interaction between objects is designed based on a disclosure protocol.

Our approach was evaluated based on two case studies. First, we carried out an analysis of the followership design on the Twitter social networking platform. We demonstrated the variability in the range of privacy utilities that a followed user and its followers can derive as a message is tweeted, retweeted or liked in the network. We then investigated a refactoring of the followership design with variants of `Consent` and `Notice`. The results showed that with a privacy objective to maximize message visibility, refactoring the Twitter followership with `Consent` did not significantly improve privacy utility. Whereas, for the same privacy objective, refactoring the design with `Notice` showed a significant improvement in privacy utility across interacting parties. The broader insight is that the design of software where user objects are associated with emergent properties and therefore changing privacy objectives needs to be adaptive privacy ready [41].

The second case study involved the scenario analysis of service-based software design patterns. Two categories involving a broker either mediating or facilitating interaction between information producers and service providers were analyzed. Our study results showed that flexibility in designing a privacy-preserving service-oriented system is also dependent on the design pattern used to mediate or facilitate interactions. These patterns include pull, push as well as their hybrid combined with or without binding. We demonstrated the strengths and weaknesses of each pattern in satisfying a privacy objective. The broader insight is the relationship between software design patterns and the satisfaction of a privacy objective.

The use of proposed technique in practice depends on two steps. First, the software designer articulates the features to be implemented using a model-driven design technique. Second, the designer defines a privacy objective to be realized in the design. This is specified in terms of the desired level of information secrecy or visibility. A subset of disclosure protocols that minimize privacy risk in the design is then proposed.

Acknowledgments

We thank the anonymous reviewers for their careful reading of our manuscript and their insightful comments and suggestions. This research is supported by the

Innovate UK CyberASAP program, EPSRC Institutional support grant award EP/P51133X/1 and EPSRC DTA.

Appendix A

Table A.1. Tenable instances in the memory of recipient and sender at t_n as a result of `Request`.

		X1 - sender generates *f* about subject		X2 – sender seeks *f* from subject	
Process		**Recipient**	$M_r(t_n)$	**Sender**	$M_s(t_n)$
request (X1)	**A1**	Unaware of *f*	*2*	Aware of *f*	*1*
	A2	Aware that the subject is aware or unaware of *f*	*3,5*	Aware that the subject is aware or unaware of *f*	*3,5*
		Aware that the sender is aware or unaware of *f*	*7,9*	Aware that the recipient is aware or unaware of *f*	*7,9*
	A3	Aware that the subject is aware or unaware that the recipient is unaware of *f*	*12,14*	Aware that the subject is aware or unaware that the sender is aware of *f*	*11,13*
		Aware that the sender is aware or unaware that the recipient is unaware of *f*	*20,22*	Aware that the recipient is aware or unaware that the sender is aware of *f*	*19,21*
	A4	Aware that the subject is aware or unaware that the sender is aware or unaware of *f*.	27,28, 29,30,	Aware that the subject is aware or unaware that the recipient is aware or unaware of *f*.	27,28, 29,30
		Aware that the sender is aware or unaware that the subject is aware or unaware of *f*.	35,36, 37,38	Aware that the recipient is aware or unaware that the subject is aware or unaware of *f*.	35,36, 37,38
request (X2)	**A1**	Unaware of *f*	*2*	Unaware of *f*	*2*
	A2	Aware that the subject is aware or unaware of *f*	*3,5*	Aware that the subject is aware of *f*	*3*
		Aware that the sender is aware or unaware of *f*	*7,9*	Aware that the recipient is aware or unaware of *f*	*7,9*
	A3	Aware that the subject is aware or unaware that the recipient is unaware of *f*	*12,14*	aware that the subject is aware or unaware that the sender is unaware of *f*	*12,14*
		Aware that the sender is aware or unaware that the recipient is unaware of *f*	*20,22*	aware that the recipient is aware or unaware that the sender is unaware of *f*	*20,22*
	A4	Aware that the subject is aware or unaware that the sender is aware or unaware of *f*.	27,28, 29,30,	Aware that the subject is aware or unaware that the recipient is aware or unaware of *f*.	27,28, 29,30
		Aware that the sender is aware or unaware that the subject is aware or unaware of *f*.	35,36, 37,38	Aware that the recipient is aware or unaware that the subject is aware or unaware of *f*.	35,36, 37,38

Table A.2. Tenable instances in the memory of sender and recipient at t_n as a result of `Sent`.

Process		**Sender**	$M_s(t_n)$	**Recipient**	$M_r(t_n)$
send	**A1**	Aware of *f*.	*1*	Aware of *f*.	*1*
	A2	Aware that the subject is aware or unaware of *f*.	*3,5*	Aware that the subject is aware or unaware of *f*.	*3,5*
		Aware that the recipient is aware or unaware of *f*.	*7,9*	Aware that the sender is aware of *f*.	*7*
	A3	Aware that the subject is aware or unaware that the sender is aware of *f*.	*11,13*	Aware that the subject is aware or unaware that the recipient is aware of *f*.	*11,13*
		Aware that the recipient is aware or unaware that the sender is aware of *f*.	*19,21*	Aware that the sender is aware or unaware that the recipient is aware of *f*.	*19,21*
	A4	Aware that the subject is aware or unaware that the recipient is aware or unaware of *f*.	27,28, 29,30	Aware that the subject is aware or unaware that the sender is aware or unaware of *f*.	27,28, 29,30
		Aware that the recipient is aware or unaware that the subject is aware or unaware of *f*.	35,36, 37,38	Aware that the sender is aware or unaware that the subject is aware or unaware of *f*.	35,36, 37,38

Table A.3. Tenable instances in the memory of subject, sender and recipient at t_n as a result of `Notice`.

Notice:s-su

a ∈ M_s			M_{su}@ t_n wrt. s		
A1	1	$A_s f$	**A2**	3	$A_{su} A_s f$
	2	$\neg A_s f$		5	$A_{su} \neg A_s f$
A2	3	$A_s A_{su} f$	**A3**	11	$A_{su} A_s A_{su} f$
	4	$\neg A_s A_{su} f$		13	$A_{su} \neg A_s A_{su} f$
	5	$A_s \neg A_{su} f$		12	$A_{su} A_s \neg A_{su} f$
	6	$\neg A_s \neg A_{su} f$		14	$A_{su} \neg A_s \neg A_{su} f$
	7	$A_s A_r f$	**A4**	27	$A_{su} A_s A_r f$
	8	$\neg A_s A_r f$		29	$A_{su} \neg A_s A_r f$
	9	$A_s \neg A_r f$		28	$A_{su} A_s \neg A_r f$
	10	$\neg A_s \neg A_r f$		30	$A_{su} \neg A_s \neg A_r f$

Notice:su-s

a ∈ M_{su}			M_s @ t_n wrt. su		
A1	1	$A_{su} f$	**A2**	3	$A_s A_{su} f$
	2	$\neg A_{su} f$		5	$A_s \neg A_{su} f$
A2	3	$A_{su} A_s f$	**A3**	11	$A_s A_{su} A_s f$
	4	$\neg A_{su} A_s f$		13	$A_s \neg A_{su} A_s f$
	5	$A_{su} \neg A_s f$		12	$A_s A_{su} \neg A_s f$
	6	$\neg A_{su} \neg A_s f$		14	$A_s \neg A_{su} \neg A_s f$
	7	$A_{su} A_r f$	**A4**	27	$A_s A_{su} A_r f$
	8	$\neg A_{su} A_r f$		29	$A_s \neg A_{su} A_r f$
	9	$A_{su} \neg A_r f$		28	$A_s A_{su} \neg A_r f$
	10	$\neg A_{su} \neg A_r f$		30	$A_s \neg A_{su} \neg A_r f$

Notice:su-r

a ∈ M_{su}			M_r @ t_n wrt. su		
A1	1	$A_{su} f$	**A2**	3	$A_r A_{su} f$
	2	$\neg A_{su} f$		5	$A_r \neg A_{su} f$
A2	3	$A_{su} A_s f$	**A4**	27	$A_r A_{su} A_s f$
	4	$\neg A_{su} A_s f$		29	$A_r \neg A_{su} A_s f$
	5	$A_{su} \neg A_s f$		28	$A_r A_{su} \neg A_s f$
	6	$\neg A_{su} \neg A_s f$		30	$A_r \neg A_{su} \neg A_s f$
	7	$A_{su} A_r f$	**A3**	11	$A_r A_{su} A_r f$
	8	$\neg A_{su} A_r f$		13	$A_r \neg A_{su} A_r f$
	9	$A_{su} \neg A_r f$		12	$A_r A_{su} \neg A_r f$
	10	$\neg A_{su} \neg A_r f$		14	$A_r \neg A_{su} \neg A_r f$

Notice:s-r

a ∈ M_s			M_r @ t_n wrt. s		
A1	1	$A_s f$	**A2**	7	$A_r A_s f$
	2	$\neg A_s f$		9	$A_r \neg A_s f$
A2	3	$A_s A_{su} f$	**A4**	35	$A_r A_s A_{su} f$
	4	$\neg A_s A_{su} f$		37	$A_r \neg A_s A_{su} f$
	5	$A_s \neg A_{su} f$		36	$A_r A_s \neg A_{su} f$
	6	$\neg A_s \neg A_{su} f$		38	$A_r \neg A_s \neg A_{su} f$
	7	$A_s A_r f$	**A3**	19	$A_r A_s A_r f$
	8	$\neg A_s A_r f$		21	$A_r \neg A_s A_r f$
	9	$A_s \neg A_r f$		20	$A_r A_s \neg A_r f$
	10	$\neg A_s \neg A_r f$		22	$A_r \neg A_s \neg A_r f$

Notice:r-su

a ∈ M_r			M_{su} @ t_n wrt. r		
A1	1	$A_r f$	**A2**	7	$A_{su} A_r f$
	2	$\neg A_r f$		8	$A_{su} \neg A_r f$
A2	3	$A_r A_{su} f$	**A3**	19	$A_{su} A_r A_{su} f$
	4	$\neg A_r A_{su} f$		21	$A_{su} \neg A_r A_{su} f$
	5	$A_r \neg A_{su} f$		20	$A_{su} A_r \neg A_{su} f$
	6	$\neg A_r \neg A_{su} f$		22	$A_{su} \neg A_r \neg A_{su} f$
	7	$A_r A_s f$	**A4**	35	$A_{su} A_r A_s f$
	8	$\neg A_r A_s f$		37	$A_{su} \neg A_r A_s f$
	9	$A_r \neg A_s f$		36	$A_{su} A_r \neg A_s f$
	10	$\neg A_r \neg A_s f$		38	$A_{su} \neg A_r \neg A_s f$

Notice:r-s

a ∈ $M_r(t_1)$			M_s @ t_n wrt. r		
A1	1	$A_r f$	**A2**	7	$A_s A_r f$
	2	$\neg A_r f$		9	$A_s \neg A_r f$
A2	3	$A_r A_{su} f$	**A4**	35	$A_s A_r A_{su} f$
	4	$\neg A_r A_{su} f$		37	$A_s \neg A_r A_{su} f$
	5	$A_r \neg A_{su} f$		36	$A_s A_r \neg A_{su} f$
	6	$\neg A_r \neg A_{su} f$		38	$A_s \neg A_r \neg A_{su} f$
	7	$A_r A_s f$	**A3**	19	$A_s A_r A_s f$
	8	$\neg A_r A_s f$		21	$A_s \neg A_r A_s f$
	9	$A_r \neg A_s f$		20	$A_s A_r \neg A_s f$
	10	$\neg A_r \neg A_s f$		22	$A_s \neg A_r \neg A_s f$

Table A.4. Tenable instances in the memory of subject and sender at t_n as a result of `Consent`.

* specific consent X3 - sender generates *f* about subject X4 - subject is the source of *f*

Process		Subject	$M_{su}(t_n)$	Sender	$M_s(t_n)$
seek Consent (X3)	**A1**	Unaware of *f*	*2*	Aware of *f*	*1*
	A2	Aware that the sender is aware of *f*·	*3*	Aware that the subject is aware or unaware of *f.*	3,5
		Aware that the recipient* is aware or unaware of *f.*	*7,9*	Aware that the recipient* is aware or unaware of *f.*	7,9
	A3	Aware that the sender is aware or unaware that the subject is unaware of *f.*	*12, 14*	Aware that the subject is aware that the sender is aware of *f.*	*11*
		Aware that the recipient* is aware or unaware that the subject is unaware of *f.*	*20, 22*	Aware that the recipient* is aware or unaware that the sender is aware of *f.*	*19, 21*
	A4	Aware that the sender is aware or unaware that the recipient* is aware or unaware of *f.*	27,28, 29,30	Aware that the subject is aware or unaware that the recipient* is aware or unaware of *f.*	27,28, 29,30
		Aware that the recipient* is aware or unaware that the sender is aware of *f.*	35,37	Aware that the recipient* is aware or unaware that the subject is aware or unaware of *f.*	35,36, 37,38
seek Consent (X4)	**A1**	Aware of *f*	1	Unaware of *f*	2
	A2	Aware that the sender is aware or unaware of *f.*	3,5	Aware that the subject is aware of *f.*	3
		Aware that the recipient* is aware or unaware of *f.*	7, 9	Aware that the recipient* is aware or unaware of *f.*	7,9
	A3	Aware that the sender is aware that the subject is aware of *f.*	11	Aware that the subject is aware or unaware that the sender is unaware of *f.*	*12,14*
		Aware that the recipient* is aware or unaware that the subject is aware of *f.*	19, 21	Aware that the recipient* is aware or unaware that the sender is unaware of *f.*	*20,22*
	A4	Aware that the sender is aware or unaware that the recipient* is aware or unaware of *f.*	27,28, 29,30	Aware that the subject is aware or unaware that the recipient* is aware or unaware of *f.*	27,28, 29,30
		Aware that the recipient* is aware or unaware that the sender is aware or unaware of *f.*	35,36, 37,38	Aware that the recipient* is aware or unaware that the subject is aware of *f.*	35,37
grant Consent (X3)	**A1**	Aware of *f*	*1*	Aware of *f*	*1*
	A2	Aware that the sender is aware of *f* .	*3*	Aware that the subject is aware of *f.*	*3*
		Aware that the recipient* is aware or unaware of *f.*	*7,9*	Aware that the recipient* is aware or unaware of *f.*	*7,9*
	A3	Aware that the sender is aware or unaware that the subject is aware of *f.*	11,13	Aware that the subject is aware that the sender is aware of *f.*	*11*
		Aware that the recipient* is aware or unaware that the subject is aware of *f.*	19,21	Aware that the recipient* is aware or unaware that the sender is aware of *f.*	*19,21*
	A4	Aware that the sender is aware or unaware that the recipient* is aware or unaware of *f.*	27,28, 29,30	Aware that the subject is aware or unaware that the recipient* is aware or unaware of *f.*	27,28, 29,30
		Aware that the recipient* is aware or unaware that the sender is aware of *f.*	35,37	Aware that the recipient* is aware or unaware that the subject is aware of *f.*	35,37
grant Consent (X4)	**A1**	Aware of *f*	1	Aware of *f*	1
	A2	Aware that the sender is aware or unaware of *f.*	3,5	Aware that the subject is aware of *f.*	3
		Aware that the recipient* is aware or unaware of *f.*	7, 9	Aware that the recipient* is aware or unaware of *f.*	7,9
	A3	Aware that the sender is aware that the subject is aware of *f.*	11	Aware that the subject is aware or unaware that the sender is aware of *f.*	*11,13*
		Aware that the recipient* is aware or unaware that the subject is aware of *f.*	19, 21	Aware that the recipient* is aware or unaware that the sender is aware of *f.*	*19,21*
	A4	Aware that the sender is aware or unaware that the recipient* is aware or unaware of *f.*	27,28, 29,30	Aware that the subject is aware or unaware that the recipient* is aware or unaware of *f.*	27,28, 29,30
		Aware that the recipient* is aware or unaware that the sender is aware or unaware of *f.*	35,36, 37,38	Aware that the recipient* is aware or unaware that the subject is aware of *f.*	35,37

Note: *Specific consent.

Table A.5. Tenable instances in the memory of subject, sender and recipient at t_n after the `Acknowledgment` of `Request`-, `Sent`-, `Notice`- and `Consent`-related processes.

X1 - sender generates *f* about subject — X2 – sender seeks *f* from subject

Process		Recipient	$M_r(t_n)$	Sender	$M_s(t_n)$
ack Request (X1)	**A2**			Aware that the recipient is unaware of *f*	*9*
	A3	Aware that the sender is aware that the recipient is unaware of *f*	*20*		
ack Request (X2)	**A2**			Aware that the recipient is unaware of *f*	*9*
	A3	Aware that the sender is aware that the recipient is unaware of *f*	*20*		

X3 - sender generates *f* about subject — X4 - subject is the source of *f*

Process		Subject	$M_{su}(t_n)$	Sender	$M_s(t_n)$
ackSeek Consent (X3)	**A1**	aware of *f*	*1*		
	A2			Aware that the subject is aware or aware of *f*.	3
	A3	Aware that the sender is aware that the subject is aware of *f*.	*11*		
	A4			Aware that the recipient* is aware or unaware that the subject is aware of *f*.	35,37
ackSeek Consent (X4)	**A3**			Aware that the subject is aware that the sender is unaware of *f*.	*12*
ackGrant Consent(X3)	**A3**	Aware that the sender is aware that the subject is aware of *f*.	11		
ackGrant Consent (X4)	**A2**	Aware that the sender is aware of *f*.	3		
	A3			Aware that the subject is aware that the sender is aware of *f*.	*11*
	A4	Aware that the recipient* is aware or unaware that the sender is aware of *f*.	35,37	Aware that the recipient* is aware or unaware that the subject is aware of *f*.	35,37

Process		Sender	$M_s(t_n)$	Recipient	$M_r(t_n)$
ack Send	**A2**	Aware that the recipient is aware of *f*.	*7*		
	A3	Aware that the recipient is aware that the sender is aware of *f*.	*19*	Aware that the sender is aware that the recipient is aware of *f*.	*19*

ackNotice:s-su						ackNotice:su-su						ackNotice:su-r					
$a \in M_s$			M_s@ t_n wrt. su			$a \in M_{su}$			M_{su} @ t_n wrt. s			$a \in M_{su}$			M_{su} @ t_n) wrt. r		
A1	1	$A_s f$	**A3**	*11*	$A_s A_{su} A_s f$	**A1**	1	$A_{su} f$	**A3**	*11*	$A_{su} A_s A_{su} f$	**A1**	1	$A_{su} f$	**A3**	*19*	$A_{su} A_r A_{su} f$
	2	$\neg A_s f$		*12*	$A_s A_{su} \neg A_s f$		2	$\neg A_{su} f$		*12*	$A_{su} A_s \neg A_{su} f$		2	$\neg A_{su} f$		*20*	$A_{su} A_r \neg A_{su} f$

ackNotice:s-r						ackNotice:r-su						ackNotice:r-s					
$a \in M_s$			M_s@ t_n wrt. r			$a \in M_r$			M_r@ t_n wrt. su			$a \in M_r(t_1)$			M_r@ t_n wrt. s		
A1	1	$A_s f$	**A3**	*19*	$A_s A_r A_s f$	**A1**	1	$A_r f$	**A3**	*11*	$A_r A_{su} A_r f$	**A1**	1	$A_r f$	**A3**	*19*	$A_r A_s A_r f$
	2	$\neg A_s f$		*20*	$A_s A_r \neg A_s f$		2	$\neg A_r f$		*12*	$A_r A_{su} \neg A_r f$		2	$\neg A_r f$		*20*	$A_r A_s \neg A_r f$

Table A.6. Process transformation overriding.

		A1			A2						A3						A4					
		su	*s*	*r*	*su*		*s*		*r*		*su*		*s*		*r*		*su*		*s*		*r*	
Process		*f*	*f*	*f*	*s*	*r*	*su*	*r*	*su*	*s*	*s*	*r*	*su*	*r*	*su*	*s*	*s*	*r*	*su*	*r*	*su*	*s*
a	**request(X1)**		1	2 {m}			3,5 {g,i,k,l}	7,9 ({n},{b,d})	3,5	7,9 {m}			11,13 {e,i,l}	19,21 {n}	12,14 ({n},{m})	20,22 ({n},{m},{b,d})			27,28, 29,30	35,36, 37,38 {f,g,i,k,l}	27,28, 29,30	35,36, 37,38
b	**ackRequest(X1)**							9 {n}								20 ({n},{m})						
c	**request(X2)**		2 {e,i,k,l,m}	2 {m}			3	7,9 ({n}, {b,d})	3,5	7,9 {m}			12,14 ({e,i,l},{m,n},{h})	20,22 ({n},{e*,i*,m})	12,14 ({n},{m})	20,22 ({n},{m} ,{b,d})			27,28, 29,30	35,36, 37,38 {f,g,i,k,l}	27,28, 29,30	35,36, 37,38
d	**ackRequest(X2)**							9 {n}								20 ({n},{m})						
e	**seekConsent(X3)**	2 {f,gi,k,l}	1		3	7*,9*	3,5 {c,e,i,k,g}	7*,9* ({n}, {b,d})			12,14 ({f,g, j,k},{i})	20*,22* {g,i,k}	11	19*,21* {n}			27,28, 29,30	35*,37*	27,28, 29,30	35*,36*, 37*,38* {f,g,i,k,l}		
f	**ack SeekConsent(X3)**	1					3				11									35*,37*		
g	**seekConsent(X4)**	1	2 {a,i,k,l,m}		3,5 {e,i,l}	7*,9*	3	7*,9* ({n}, {b,d})			11	19*,21*	12,14 ({i,l},{a,k,m,n})	20*,22* ({n},{a,i*,m})			27,28, 29,30	35*,36*, 37*,38* {i,l}	27,28, 29,30	35*,37*		
h	**ack SeekConsent(X4)**												12 ({i,l},{a,k,m,n})									
i	**grantConsent(X3)**	1	1		3	7*,9*	3	7*,9* ({n}, {b,d})			11,13 {f,g,j,k}	19*,21*	11	19*,21* {n}			27,28, 29,30	35*,37*	27,28, 29,30	35*,37*		
j	**ack GrantConsent(X3)**										11											
k	**grantConsent(X4)**	1	1		3,5 {e,i,l}	7*,9*	3	7*,9* ({n}, {b,d})			11	19*,21*	11,13 {e,i,l}	19*,21* {n}			27,28, 29,30	35*,36*, 37*,38* {i,l}	27,28, 29,30	35*,37*		
l	**ack GrantConsent(X4)**				3								11					35*,37		35*,37*		
m	**send**		1	1			3,5 {c,e,g,i,k}	7,9 ({n}, {b,d})	3,5	7			11,13 {e,i,l}	19,21 {n}	11,13 {n}	19,21 {n}			27,28, 29,30	35,36, 37,38 {f,g,i,k,l}	27,28, 29,30	35,36, 37,38
n	**ackSend**						3,5 {c,e,g,i,k}	7					11,13 {e,i,l}	19	11,13 {n}	19						

Note: $a\{x,\ldots\}$: Here, a is overridden by the tenable instance(s) resulting from either x or $(\ldots)$.
$a\ (\{x,\ldots\},\{y,\ldots\})$: Here, a is overridden by tenable instance(s) resulting from either x or $(\ldots)$. otherwise by tenable instance(s) resulting from either y or $(\ldots)$.

References

1. S. Abu-Nimeh, S. Miyazaki and N. R. Mead, Integrating privacy requirements into security requirements engineering, in *Proc. Twenty-First Int. Conf. Software Engineering and Knowledge Engineering*, 2009, pp. 542–547.
2. R. Anderson and M. Tyler, The economics of information security, *Science* **314**(5799) (2006) 610–613.
3. I. Adjerid, A. Acquisti, L. Brandimarte and G. Loewenstein, Sleights of privacy: Framing, disclosures, and the limits of transparency, in *Proc. Ninth Symp. Usable Privacy and Security*, 2013.
4. P. Anthonysamy, P. Greenwood and A. Rashid, Social networking privacy: Understanding the disconnect from policy to controls, *Computer* **46**(6) (2013) 60–67.
5. P. Anthonysamy and A. Rashid, Software engineering for privacy in-the-large, in *Proc. 2015 IEEE/ACM 37th IEEE Int. Conf. Software Engineering*, 2015.
6. G. Aucher, B. Guido and L. van der Torre, Privacy policies with modal logic: The dynamic turn, in *Proc. Int. Conf. Deontic Logic in Computer Science*, 2010.
7. A. Barth, D. Anupam, M. John and H. Nissenbaum, Privacy and contextual integrity: Framework and applications, in *Proc. 2006 IEEE Symp. Security and Privacy*, 2006.
8. M. Benantar, *Access Control Systems: Security, Identity Management and Trust Models* (Springer Science & Business Media, 2006).
9. K. Bernsmed, Applying privacy by design in software engineering: An European perspective, in *Proc. Second Int. Conf. Advances and Trends in Software Engineering*, 2016.
10. D. Bonnay and P. Egre, Inexact knowledge with introspection, *J. Philos. Log.* **38**(2) (2009) 179–227.
11. D. Breaux, H. Hanan and R. Ashwini, Eddy, a formal language for specifying and analyzing data flow specifications for conflicting privacy requirements, *Requir. Eng.* **19**(3) (2014) 281–307.
12. D. Budgen, *Software Design* (Pearson Education, 2003).
13. G. Calikli, M. Law, A. Bandara, A. Rusoo, L. W. F. Dickens, B. Price, A. Stuart, M. Levine and B. Nuseibeh, Privacy dynamics: Learning privacy norms for social software, in *Proc. 11th Int. Symp. Software Engineering for Adaptive and Self-Managing Systems*, 2016.
14. A. Cavoukian, Privacy by design: The 7 foundational principles, Document, Information and Privacy Commissioner of Ontario, Canada (2009).
15. A. Cavoukian, Operationalizing privacy by design: A guide to implementing strong privacy practices, Document, Information and Privacy Commissioner of Ontario, Canada (2012).
16. B. Curtis, H. Krasner and N. Iscoe, A field study of the software design process for large systems, *Commun. ACM* **31**(11) (1988) 1268–1287.
17. H. van Ditmarsch, T. French and F. Velazquez-Quesada, Action models for knowledge and awareness, in *Proc. 11th Int. Conf. Autonomous Agents and Multiagent Systems*, Vol. 2, 2012, pp. 1091–1098.
18. OASIS, MQTT Version 3.1.1 Plus Errata 01, OASIS Standard Incorporating Approved Errata 01 (2015).
19. T. Eugster, P. Felber, R. Guerraoui and A. Kermarrec, The many faces of publish/subscribe, *ACM Comput. Surv.* **35**(2) (2003) 114–131.
20. R. Fagin and J. Halpern, Belief, awareness, and limited reasoning, *Artif. Intell.* **34**(1) (1987) 39–76.
21. B. Friedman, P. Lin and J. K. Miller, Informed consent by design, in *Security and Usability* (O'Reilly Media, 2001), pp. 495–521.

22. T. Gilb and S. Finzi, *Principles of Software Engineering Management* (Addison-Wesley, 1988).
23. M. Green and M. Smith, Developers are not the enemy!: The need for usable security APIs, *IEEE Secur. Priv.* **14**(5) (2016) 40–46.
24. S. Gürses, T. Carmela and D. Claudia, Engineering privacy by design reloaded, in *Proc. Amsterdam Privacy Conf.*, 2015.
25. W. Hartzog, *Privacy's Blueprint: The Battle to Control the Design of New Technologies* (Harvard University Press, 2018).
26. Q. He and A. Antón, A framework for modeling privacy requirements in role engineering, in *Proc. Ninth Int. Workshop Requirements Engineering: Foundations of Software Quality*, 2003.
27. A. Heifetz, M. Meier and B. Schipper, Interactive unawareness, *J. Econ. Theory* **130**(1) (2006) 78–94.
28. J. Hintikka, *Knowledge and Belief: An Introduction to the Logic of the Two Notions* (Cornell University Press, Ithaca, 1965).
29. H. Hoepman, Privacy design strategies, in *Proc. IFIP Int. Information Security Conf.*, 2014.
30. N. Huhns and M. Singh, Service-oriented computing: Key concepts and principles, *IEEE Internet Comput.* **9**(1) (2005) 75–81.
31. GitHub, JAX-RS specification (2019), https://github.com/jax-rs.
32. GitHub, JAX-RS specification using Jersey (2019), https://jersey.github.io/.
33. Oracle, Java Message Service (2019), https://docs.oracle.com/cd/E19957-01/816-5904-10/816-5904-10.pdf.
34. R. Jennings, On agent-based software engineering, *Artif. Intell.* **117**(2) (2000) 277–296.
35. N. R. Mead, Benefits and challenges in the use of case studies for security requirements engineering methods, in *Security-Aware Systems Applications and Software Development Methods* (IGI Global, 2012), pp. 89–107.
36. J. McGovern, S. Tyagi, M. Stevens and S. Mathew, *Java Web Services Architecture* (Elsevier, 2003).
37. J. Ransome and A. Misra, *Core Software Security: Security at the Source* (Auerbach, Boston, 2013).
38. H. Nissenbaum, *Privacy in Context: Technology, Policy, and the Integrity of Social Life* (Stanford University Press, 2009).
39. H. Nissenbaum, A contextual approach to privacy online, *Daedalus* **140**(4) (2011) 32–48.
40. I. Omoronyia, The case for privacy awareness requirements, *Int. J. Secur. Softw. Eng.* **7**(2) (2016) 19–36.
41. I. Omoronyia, L. Cavallaro, M. Salehie, L. Pasquale and B. Nuseibeh, Engineering adaptive privacy: On the role of privacy awareness requirements, in *Proc. Int. Conf. Software Engineering*, 2013.
42. M. P. Papazoglou, P. Traverso, S. Dustdar and F. Leymann, Service-oriented computing: State of the art and research challenges, *Computer* **40**(11) (2007) 38–45.
43. S. Petronio, Communication privacy management theory, in *The International Encyclopedia of Interpersonal Communication*, Vol. 1 (Wiley-Blackwell, 2015), pp. 353–360.
44. A. Poller, K. Laura, T. Sven, E. Felix and K. Katharina, Can security become a routine?: A study of organizational change in an agile software development group, in *Proc. Computer Supported Cooperative Work and Social Computing*, 2017.
45. NOKIA, Privacy Engineering and Assurance: The emerging engineering discipline for implementing privacy by design (2014), https://iapp.org/media/pdf/resource_center/Privacy_Engineering+assurance-Nokia_9-14.pdf.

46. Microsoft, Privacy guidelines for developing software products and services (2008), https://download.microsoft.com/download/9/3/5/935520EC-D9E2-413E-BEA7-0B865A79B18C/Privacy%20in%20Software%20Development.ppsx.
47. J. Sabo and M. Willett, OASIS privacy management reference model and methodology (PMRM) Version 1.0, OASIS Committee Specification 2 (2013).
48. D. C. Schmidt, M. Fayad and R. E. Johnson, Software patterns, *Commun. ACM* **39**(10) (1996) 37–39.
49. E. Semino, Book Review: Possible Worlds, Artificial Intelligence and Narrative Theory, *Lang. Lit.* **2**(2) (1993) 146–148.
50. R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman and D. Ward, Gryphon: An information flow based approach to message brokering, preprint (1998), arXiv:cs/9810019 [cs.DC].
51. Twitter, Twitter Help Manual (2019), https://help.twitter.com/en/using-twitter.
52. Twitter, Twitter Help Manual (2019), http://activemq.apache.org.
53. A. Whitten and D. Tygar, Why Johnny can't encrypt: A usability evaluation of PGP 5.0, in *Proc. USENIX Security Symp.*, 1999.
54. D. Wright and P. De Hert, Introduction to privacy impact assessment, in *Privacy Impact Assessment* (Springer, Dordrecht, 2012), pp. 3–32.
55. T. Williamson, *Knowledge and Its Limits* (Oxford University Press, 2002).
56. S. S. Yau and J. J.-P. Tsai, A survey of software design techniques, *IEEE Trans. Softw. Eng.* **SE-12** (1986) 713–721.
57. I. Yevseyeva, C. Morisset and A. van Moorsel, Modeling and analysis of influence power for information security decisions, *Perform. Eval.* **98** (2016) 36–51.
58. H. Ziegeldorf, M. Oscar and W. Klaus, Privacy in the Internet of Things: Threats and challenges, *Secur. Commun. Netw.* **7**(12) (2014) 2728–2742.