Capacity: an Abstract Model of Control over Personal Data
Daniel Le Métayer, Pablo Rauzy

To cite this version:

HAL Id: hal-01939847
https://hal.archives-ouvertes.fr/hal-01939847
Submitted on 29 Nov 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Capacity: an Abstract Model of Control over Personal Data

Daniel Le Métayer
Inria, Université de Lyon
daniel.le-metayer@inria.fr

Pablo Rauzy∗
Université Paris 8 / LIASD
pablo.rauzy@univ-paris8.fr

ABSTRACT

While the control of individuals over their personal data is increasingly seen as an essential component of their privacy, the word “control” is usually used in a very vague way, both by lawyers and by computer scientists. This lack of precision may lead to misunderstandings and makes it difficult to check compliance. To address this issue, we propose a formal framework based on capacities to specify the notion of control over personal data and to reason about control properties. We illustrate our framework with social network systems and show that it makes it possible to characterize the types of control over personal data that they provide to their users and to compare them in a rigorous way.

KEYWORDS
privacy, control, formal model

ACM Reference Format:

1 INTRODUCTION

Instead of the “right to be let alone”, as originally coined by Samuel Warren and Louis Brandeis in their landmark article [16], privacy is increasingly seen as the ability for individuals to control their personal data. The current trend is also to recommend the integration of privacy requirements in the earliest stages of the design of a product, following the privacy by design approach. However, even if the notions of privacy as control and privacy by design are predominant in the privacy literature, clear definitions of their meanings are still missing. The word “control” in particular is usually used in a very vague way in this context, both by lawyers and by computer scientists. This lack of precision may lead to misunderstandings and makes it difficult to check compliance. To address this issue, we propose a formal framework to specify the notion of control over personal data and to reason about control properties.

1 INTRODUCTION

Instead of the “right to be let alone”, as originally coined by Samuel Warren and Louis Brandeis in their landmark article [16], privacy is increasingly seen as the ability for individuals to control their personal data. The current trend is also to recommend the integration of privacy requirements in the earliest stages of the design of a product, following the privacy by design approach. However, even if the notions of privacy as control and privacy by design are predominant in the privacy literature, clear definitions of their meanings are still missing. The word “control” in particular is usually used in a very vague way in this context, both by lawyers and by computer scientists. This lack of precision may lead to misunderstandings and makes it difficult to check compliance. To address this issue, we propose a formal framework to specify the notion of control over personal data and to reason about control properties.

The notion of control occurs in different contexts such as “access control” or “usage control” in computer science, but none of these variants really encapsulates the intuition underlying the notion of control over personal data. Previous work [8] has identified three dimensions of control over personal data corresponding to the capacities for an individual:

(1) to perform actions on their personal data,
(2) to prevent others from performing actions on their personal data, and
(3) to be informed of actions performed by others on their personal data.

Actions can be of various kinds including, without limitation, consultation, modification, deletion and disclosure. In this paper, we build on this reflection to define a formal model of control and we show its relevance through the description of different options of implementation of a social network system. We show that each option provides a different type of control that can be characterized in a formal way.

Contributions and organization of the paper. In Section 2, we introduce an abstract model, called Capacity, which makes it possible to express, inter alia, the three capacities put forward in [8]. We proceed in Section 3 with the definition of requirements characterizing typical variants of control: action control, observability control, authorization control, and notification control. In Section 4, we introduce a concrete case study, a social network system, with its specification and describe it within the Capacity model. Section 5 presents three implementations of the specifications of the case study corresponding to different architectural choices (respectively centralized, peer to peer, and federated). We prove that they meet different control requirements in the Capacity model. In Section 6, we present previous works on control and compare them with our model before suggesting avenues for further research in Section 7. In Appendix B, we define an order relation which is useful to compare different systems based on the level of control that they provide (but not necessary to follow the body of the paper).

2 CAPACITY

The goal of the Capacity model is to make it possible to express, in a very general way, the three dimensions of controls (the capacities) introduced in Section 1 and to use them as a basis for reasoning about control. The guiding principles for the design of Capacity were therefore abstraction and minimality. Basically, the model is based on a set of agents that can perform operations on resources. These operations can be constrained by control requirements which form the core of the model. In this section, we first introduce the building blocks of the model in Sections 2.1 through 2.3 before defining the notion of requirement and its semantics in Section 2.4.
2.1 Objects
The Capacity model is based on four types of atomic objects drawn from finite distinct sets \(\mathcal{A}, \mathcal{R}, \mathcal{O}\) and \(\mathcal{C}\):

- **Agents**, noted \(a_1, a_2, \ldots \in \mathcal{A}\), represent active entities (typically users or services).
- **Resources**, noted \(r_1, r_2, \ldots \in \mathcal{R}\), typically include personal data.
- **Operations**, noted \(o_1, o_2, \ldots \in \mathcal{O}\), may typically include access, update, deletion, and communication operations.
- **Contexts**, noted \(c_1, c_2, \ldots \in \mathcal{C}\), denote the context in which an agent operates on a resource, i.e., any external factors relevant to the operation. Depending on the application, the context can include information such as location, time, or relationships between agents. Contexts can be used in particular to distinguish successful applications of an operation to the same arguments, or more high-level concepts such as the purpose for which personal data are processed.

2.2 Actions
We call an action the application of an operation to a list of parameters:

\[
\alpha \triangleq o(x_1, \ldots, x_n)
\]

the action consisting of the application of operation \(o\) to arguments \(x_1, \ldots, x_n\) in context \(c\). The arguments \(x_i\) can be resources or agents. When the context is irrelevant, we omit the context and simply write \(o(x_1, \ldots, x_n)\). \(\Delta\) denotes the set of actions.

2.3 Relations
To be able to express privacy requirements, we need to introduce three relations on atomic objects:

- **Pers(r, a)** expresses that resource \(r\) is a personal data\(^3\) of agent \(a\).
- **Intr(r, a)** means that resource \(r\) is involved\(^4\) in action \(\alpha\), and
- **Trust(a, b)** expresses that agent \(a\) trusts agent \(b\).

The last relation can be useful to distinguish situations in which the control of an agent over their personal data depends only on trusted agents from situations in which it depends also on untrusted agents\(^5\).

2.4 Requirements
We define a requirement \(R\) as a relation \(\text{Can}^R(a, \alpha, E, W) \subseteq \mathcal{A} \times \mathcal{R} \times \mathcal{P}(\mathcal{A}) \times \mathcal{P}(\mathcal{A})\) such that \(a \notin E\) and \(a \notin W\). The intuition for \(\text{Can}^R(a, \alpha, E, W)\) is that agent \(a\) can perform action \(\alpha\) only if this action is enabled by all agents in \(E\) (the enablers), while all agents in \(W\) (the witnesses) have to be informed about the performance of this action by \(a\). In other words, agents in \(E\) can prevent \(a\) from doing \(\alpha\).

By convention, we use \(\bot\) to denote an undefined or phantom agent, which never performs nor enables any action. Therefore, \(\text{Can}^R(a, \alpha, \{\bot\}, W)\) expresses the fact that requirement \(R\) prevents \(a\) from performing action \(\alpha\). Conversely, \(\text{Can}^R(a, \alpha, \emptyset, W)\) expresses the fact that requirement \(R\) unconditionally allows \(a\) to perform action \(\alpha\).

Requirements make it possible to express the three capacities mentioned in the introduction, depending on the position of an agent \(x\) in the parameters of \(\text{Can}^R(a, \alpha, E, W)\):

1. a property with \(x = a\) expresses the conditions under which \(x\) has the capacity to perform an action,
2. a property with \(x \in E\) expresses the capacity of \(x\) to prevent others from performing an action, and
3. a property with \(x \in W\) expresses the capacity of \(x\) to be informed of the performance of an action by another agent.

We further elaborate on these options in Section 3 in which we take a systematic approach and describe four types of control.

**Definition 1** (Compact requirements). A requirement \(R\) is said to be compact if \(\forall a \in \mathcal{A}, \forall \alpha \in \mathcal{A}, \forall E, W \subseteq \mathcal{A}, \text{Can}^R(a, \alpha, E, W) \land \text{Can}^R(a, \alpha, E, W') \Rightarrow E = E' \land W = W'\).

Without loss of generality, we only consider compact requirements in the rest of this paper, and to improve readability, we introduce the following functions:

- \(\text{Can}^R_{\bot}(a, \alpha) = E\) if \(\exists W \subseteq \mathcal{A}\) s.t. \(\text{Can}^R(a, \alpha, E, W)\),
- \(\text{Can}^R_{\bot}(a, \alpha) = W\) if \(\exists E \subseteq \mathcal{A}\) s.t. \(\text{Can}^R(a, \alpha, E, W)\),

which are well-defined because of the restriction to compact requirements.

In order to define the semantics of requirements, we characterize execution traces \(\theta\) in a very abstract way (the type of \(\theta\) remains opaque at this level of abstraction), in terms of the following properties:

1. \(\theta \triangleright \text{Requests}(a, \alpha)\) means that, in trace \(\theta\), agent \(a\) attempts to perform action \(\alpha\).
2. \(\theta \triangleright \text{Enables}(a, b, \alpha)\) means that, in trace \(\theta\), agent \(a\) enables the performance of action \(\alpha\) by agent \(b\).
3. \(\theta \triangleright \text{Does}(a, b, \alpha)\) means that, in trace \(\theta\), agent \(a\) performs action \(\alpha\) on behalf of agent \(b\). Does makes it possible to distinguish the agent actually performing an action from the agent that has initiated this action, which is useful in many situations.
4. \(\theta \triangleright \text{Notifies}(a, b, c, \alpha)\) means that, in trace \(\theta\), agent \(a\) notifies to agent \(b\) the performance of action \(\alpha\) on behalf of agent \(c\).

At this stage, we are content with an intuitive description of the above properties which are used below to define trace consistency and trace compliance. For each implementation, these properties will be defined precisely in terms of the corresponding execution traces (Section 5). It should be noted that this abstract level does not involve any notion of time or position in a trace: as suggested above, the context can be used to distinguish actions corresponding to different occurrences of application of an operation. At the implementation level, this information can be refined, for example, in terms of the position of an event in the trace.

\(^{\text{a}}\)It should be noted that \(o_\alpha(x_1, \ldots, x_n)\) is not the result of the application of \(o_\alpha\) to parameters \(x_1, \ldots, x_n\). Formally speaking, it is just a convenient notation for the tuple \([o_\alpha, c, x_1, \ldots, x_n]\).

\(^{\text{b}}\)A resource can be the personal data of multiple agents. Therefore, we can have \(\text{Pers}(r, a_1)\) and \(\text{Pers}(r, a_2)\) with \(a_1 \neq a_2\).

\(^{\text{c}}\)More precisely, \(r\) is a parameter (or is included in one) of the operation of \(\alpha\).

\(^{\text{d}}\)The notion of trust is intentionally left informal at our level of abstraction.
Definition 2 (Trace consistency). Trace $\theta$ is said to be consistent if 
$\theta \vdash \text{Does}(c, a, a) \implies \theta \vdash \text{Requests}(a, a)$, and $\theta \vdash \text{Notifies}(b, c, a) \implies \exists d, \theta \vdash \text{Does}(d, c, a)$

Definition 2 expresses the fact that a trace is inconsistent if it includes an action performed on behalf of an agent that has not requested it or the notification of an action that has not been performed.

Definition 3 (Trace completeness). Trace $\theta$ is said to be complete with respect to requirement $R$ if $\theta \models R$ if and only if $\forall a \in A, \forall b \in A, \forall c \in A, \theta \vdash \text{Does}(c, a, a)$, $\theta \vdash \text{Enables}(b, a, a)$, and $\forall b \in \text{Can}_R(a, a), \exists c \in A, \theta \vdash \text{Does}(c, b, a)$

Definition 3 characterizes a complete trace by the fact that a requested action is always performed if all the enabling agents have actually enabled it.

In the rest of this paper, we assume that traces are both complete and consistent.

We can now characterize the notion of compliance of a trace with a requirement.

Definition 4 (Trace compliance). Trace $\theta$ is compliant with requirement $R$, noted $\theta \models R$ if and only if $\forall a \in A, \forall b \in A, \forall c \in A, \theta \vdash \text{Does}(c, a, a)$, $\theta \vdash \text{Enables}(b, a, a)$, and $\forall b \in \text{Can}_R(a, a), \exists c \in A, \theta \vdash \text{Notifies}(c, b, a, a)$.

In a nutshell, trace $\theta$ complies with requirement $R$ if all Can$^R$ constraints are met by $\theta$: no action is performed unless it is enabled by all its enablers and all agents that have to be notified are notified.

3 TYPES OF CONTROL

Capacity requirements can be used to characterize different forms and levels of control which can typically be required or expected by data subjects. In the following, we introduce four types of control:

- **Action control**, characterizing an agent’s control on the actions that it initiates.
- **Observability control**, characterizing an agent’s capacity to perform actions that are not observable by others.
- **Authorization control**, characterizing an agent’s control on the actions initiated by others.
- **Notification control**, characterizing an agent’s capacity to be informed about actions performed by others.

For each type of control, we distinguish two levels: absolute control and relative control (in addition to level zero, or lack of control).

Definition 5 (Action control). Requirement $R$ provides agent $a$ an absolute action control over action $\alpha$, noted $\text{AA}_R(a, \alpha)$, if and only if $\text{Can}_R(a, \alpha) = \emptyset$.

Requirement $R$ provides agent $a$ a relative action control over action $\alpha$, noted $\text{RA}_R(a, \alpha)$, if and only if $\forall b \in A, b \in \text{Can}_R(a, \alpha) \implies \text{Trust}(a, b)$.

$\text{AA}_R(a, \alpha)$ means that $a$ can perform action $\alpha$ without needing any enabler. In other words, $a$ does not depend on any other agent to perform $\alpha$. In contrast, $\text{RA}_R(a, \alpha)$ means that $a$ depends on other agents to perform $\alpha$ but all these agents are trusted by $a$.

Definition 6 (Observability control). Requirement $R$ provides agent $a$ an absolute observability control over action $\alpha$, which is noted $\text{AO}_R(a, \alpha)$, if and only if $\text{Can}_R(a, \alpha) = \emptyset$.

Requirement $R$ provides agent $a$ a relative observability control over action $\alpha$, which is noted $\text{RO}_R(a, \alpha)$, if and only if $\forall b \in A, b \in \text{Can}_R(a, \alpha) \implies \text{Trust}(a, b)$.

$\text{AO}_R(a, \alpha)$ means that $a$ can perform $\alpha$ discreetly, that is to say without being observable by other agents. In contrast, $\text{RO}_R(a, \alpha)$ means that other agents can know that $a$ performs $\alpha$ but all these agents are trusted by $a$.

We should emphasize that we consider only observability of actions here and do not express other forms of observability or notions of explicit information flows for example.

Definition 7 (Authorization control). Requirement $R$ provides agent $a$ an absolute authorization control over action $\alpha$, which is noted $\text{AH}_R(a, \alpha)$, if and only if $\forall b \in A$ such that $b \neq a$, $\text{Can}_R(b, a) = \{a\}$.

Requirement $R$ provides agent $a$ a relative authorization control over action $\alpha$, which is noted $\text{RH}_R(a, \alpha)$, if and only if $\forall b \in A$ such that $b \neq a$, $a \in \text{Can}_R(b, a)$.

$\text{AH}_R(a, \alpha)$ means that $a$ can enable the performance of action $\alpha$ by other agents and is the only agent having this power. In other words, the possibility for another agent to perform $\alpha$ depends only on $a$. In contrast, $\text{RH}_R(a, \alpha)$ means that $a$ is not the only agent having this power. In other words, the possibility for another agent to perform $\alpha$ depends not only on $a$ but also on other agents.

Definition 8 (Notification control). Requirement $R$ provides agent $a$ a notification control over action $\alpha$, noted $\text{AN}_R(a, \alpha)$, if and only if $\forall b \in A$ such that $b \neq a$, $\text{Can}_R(b, a) = \{a\}$.

Requirement $R$ provides agent $a$ a relative notification control over action $\alpha$, noted $\text{RN}_R(a, \alpha)$, if and only if $\forall b \in A$ such that $b \neq a$, $a \in \text{Can}_R(b, a)$.

$\text{AN}_R(a, \alpha)$ means that $a$ is informed about the performance of action $\alpha$ by other agents and is the only agent having this power. In contrast, $\text{RN}_R(a, \alpha)$ means that $a$ is not the only agent having this power.

Extensions. All the above definitions of control can be generalized to personal data and agents. For example:

- Requirement $R$ provides agent $a$ an absolute action control over $r$, noted $\text{AA}_R(a, r)$, if $\forall a \in A$ s.t. $\text{In}(r, a)$, then $\text{AA}_R(a, r)$.
- Requirement $R$ provides agent $a$ an absolute action control over their personal data, noted $\text{AA}_R(a)$, if $\forall r \in R$, $\text{Pers}(r, a)$ $\implies$ $\text{AA}_R(a, r)$.

It is easy to check that, for each variant, absolute control implies relative control. Considering that action, observability, authorization, and notification are four independent variants of control, the above definitions give rise to a lattice (using the order defined by implication) made of 81 forms of control for each action, data, or agent.

---

1For the sake of simplicity, we consider that an agent requesting an action cannot change their mind and cancel the request before the actual performance of the action.

2We recall that $a \in \text{Can}_R(b, a)$ means that $a$ has the capacity to prevent $b$ from performing $\alpha$.

3Which is equal to $3^3$ considering that 3 levels are possible for each variant: absolute control, relative control, and lack of control.
4 SOCIAL NETWORK SYSTEM

The goal of the Capacity model presented in the previous sections is to capture the meaning of the notion of control in a very abstract and general way. In the rest of this paper, we describe the application of this model to a specific case study and show its relevance to assess different implementation choices. We choose a social network system (“SNS” in the sequel) to illustrate the model, not only because of the central role of social networks nowadays but also because they raise significant challenges in terms of control. We first introduce generic definitions allowing us to describe an SNS in the Capacity framework in Section 4.1 and define some requirements that have to be met by any SNS implementation. Then, in Section 5, we respectively describe a centralized SNS implementation, a peer to peer SNS implementation, and a federated SNS implementation, and we show that they provide different types of control.

4.1 Generic Definitions

Due to space considerations, we focus on the following set of core SNS features in this paper: profile update, access to profiles, and connection with other users.

The first step to apply the Capacity framework is to define the sets \( \mathcal{A} \) (agents), \( \mathcal{R} \) (resources), \( \mathcal{O} \) (operations) and \( \mathcal{C} \) (contexts) introduced in Section 2, as well as the relations \( \text{Pers} \), \( \text{Trust} \) and \( \text{In} \) introduced in the same section.

- \( \mathcal{A} \) includes the users of the social network, noted \( u_i \). Depending on the implementation, \( \mathcal{A} \) may also include services such as the SNS agent itself (noted \( sn \)). In order to distinguish users from other agents, we introduce a unary relation \( \text{User} \) defined as \( \text{User}(x) \iff \exists i \in \mathbb{N}, x = u_i. \)
- \( \mathcal{R} \) includes two resources per user \( u_i \): their name, noted \( n_i \), and their complete profile, noted \( p_i \) (which includes \( n_i \)).
- \( \mathcal{O} \) consists of the operations update-profile, access-profile, and connect:
  - \( \text{update-profile}_c(u_i) \) is the update of the profile \( p_i \) of user \( u_i \) in context \( c \),
  - \( \text{access-profile}_c(u_i) \) is the access to the profile \( p_i \) of user \( u_i \) in context \( c \),
  - \( \text{connect}_c(u_i) \) is the connection to user \( u_i \) in context \( c \). Note that an operation does not refer to the agent performing it. This information is provided by relations (such as \( \text{Can}, \text{Requests}, \text{Enables} \)), in which actions involving the operation appear.
- \( \mathcal{C} \) is defined by \( \mathcal{C} \subseteq \mathbb{N} \). A context is simply a natural number corresponding to an index in an execution trace (allowing for disambiguation of otherwise similar events).
- \( \text{Pers} \) is defined by \( \text{Pers}(r, a) \iff a = u_i \land (r = n_i \lor r = p_i) \), assuming for the sake of simplicity, that agents do not share personal data.
- \( \text{Trust} \) can take different values depending on the agents. For example, some users may trust the SNS agent while others do not, some users may trust some peers but not all other users, etc.
- \( \text{In} \) is derived from the definition of \( \mathcal{O} \):

\[
\text{In}(r, a) \iff (a = \text{update-profile}_c(u_i) \land (r = p_i \lor r = n_i)) \\
\lor (a = \text{access-profile}_c(u_i) \land (r = p_i \lor r = n_i)) \\
\lor (a = \text{connect}_c(u_i) \land r = n_i).
\]

4.2 Generic SNS Requirements

Some properties, which can be seen as the control oriented part of the SNS specification, have to be met by any SNS implementation. The most important generic requirements are the following:

1. A user cannot prevent another user to update or access their own profile:
   \[
   \forall u_i, u_j \in \mathcal{A}, \text{User}(u_j) \land \text{User}(u_i) \land u_i \neq u_j \\
   \implies u_j \notin \text{Can}^R(u_i, \text{update-profile}_c(u_i)) \\
   \land u_j \notin \text{Can}^R(u_i, \text{access-profile}_c(u_i)).
   \]
2. A user cannot update the profile of another user:
   \[
   \forall u_i, u_j \in \mathcal{A}, \text{User}(u_j) \land \text{User}(u_i) \land u_i \neq u_j \\
   \implies \bot \in \text{Can}^R(u_i, \text{update-profile}_c(u_j)).
   \]
3. A user can always refuse a connection request from another user:
   \[
   \forall u_i, u_j, u_k \in \mathcal{A}, u_i \neq u_j \neq u_k \neq u_i, \\
   \land \text{User}(u_i) \land \text{User}(u_j) \land \text{User}(u_k) \\
   \implies u_k \notin \text{Can}^R(u_i, \text{connect}_c(u_j)) \\
   \land u_k \notin \text{Can}^R(u_j, \text{connect}_c(u_j)).
   \]

Other properties which are not used in this paper\(^{11}\) are not included in the above list for the sake of conciseness.

5 COMPARING THREE SNS IMPLEMENTATIONS

In this section we describe three SNS implementations: one centralized, one peer to peer, and one federated, and we show that they provide different types of control.

5.1 Centralized SNS Implementation

As a first example of architectural choice, we consider in this section the most common option, that is a centralized SNS implementation. This implementation involves, in addition to user agents \( u_i \), the SNS agent \( sn \).

In order to describe this implementation in the Capacity framework, we characterize its execution traces in Section 5.1.1 and define the \( \text{Requests}(a, c), \text{Enables}(a, b, c) \), and \( \text{Does}(a, b, c) \) trace properties\(^{12}\) in Section 5.1.2. Then, we can establish the types and levels of control provided by this implementation in Section 5.1.3.

5.1.1 Execution Traces. Concrete traces in the centralized implementation are sequences of the following events. As a convention, events starting with “U” are those initiated by a user agent \( u_i \) and those starting with “S” are initiated by \( sn \).

\(^{11}\)For example, users cannot refuse access to their profiles to users that are connected to them.
\(^{12}\)Due to lack of space, we chose to focus on certain aspects of control and thus voluntarily omit \( \text{Notifies}(a, b, c, d) \).
• $U\text{-req-upd-profile}(u_i, p)$: $u_i$ sends an update $p$ of their profile $p_i$ to $sn$.
• $S\text{-do-upd-profile}(u_i, p)$: $sn$ sets $u_i$’s profile to $p$.
• $U\text{-req-acc-profile}(u_i, u_j)$: $u_i$ sends to $sn$ a request to access $u_j$’s profile.
• $S\text{-do-acc-profile}(u_i, u_j)$: $sn$ grants $u_i$’s request to access $u_j$’s profile.
• $U\text{-req-conn}(u_i, u_j)$: $u_i$ sends to $sn$ a request to be connected to $u_j$.
• $S\text{-transfer-req-conn}(u_i, u_j)$: $sn$ forwards to $u_j$ the connection request from $u_i$.
• $U\text{-accept-req-conn}(u_i, u_j)$: $u_i$ accepts $u_j$’s connection request.
• $U\text{-reject-req-conn}(u_i, u_j)$: $u_i$ rejects $u_j$’s connection request.
• $S\text{-do-conn}(u_i, u_j)$: $sn$ sets up the connection between $u_i$ and $u_j$.

In the following, we note $\theta_n$ the $n$th event of an execution trace $\theta$. The above events cannot occur in any order in a valid trace. Space considerations prevent us from presenting the full definition of valid C-traces in the core of the paper. The interested reader can find it in Appendix A (Definition 12). To follow the paper, it is sufficient to understand that in a valid C-trace $sn$ does not act spontaneously, in particular no action can be performed on behalf of an agent if this agent has not previously requested this action.

5.1.2 Trace Properties. In order to establish the control requirements provided by the centralized implementation as defined in Section 3, we first have to define the trace properties: $Requests(a, \alpha)$, $Enables(a, b, \alpha)$, and $Does(a, b, \alpha)$, which means that these properties are false in all other cases. The context $n$ associated with an action is defined as the index in the trace when this action was requested. Remark that the same event may implement several properties: e.g., event $S\text{-do-acc-profile}(u_i, u_j)$ implements both $Enables(sn, u_i, access-profile_p(u_j))$ and $Does(sn, u_i, access-profile_p(u_j))$ at the same time because granting profile access and providing profile is implemented in a single step by $sn$ in this architecture.

5.1.3 Control Properties. As discussed in Section 2 and 3, control can be considered from different perspectives (action, observability, authorization, and notification) and analyzed for each agent and with respect to each action. For the sake of conciseness, we focus on two types of control here:

• Action control of users $u_i$ over the update of their profiles.
• Authorization control of users $u_i$ over the connections to their profile (requested by other users).

Thus we introduce the following control requirement.

**Definition 9** ($Rc$ Requirement). Requirement $Rc$ is defined by:

1. $Can_{Rc_{\alpha}}(u_i, update-profile_p(u_i)) = \{sn\}$
2. $Can_{Rc_{\alpha}}(u_i, connect_p(u_j)) = \{sn, u_j\}$

All other sets are considered empty, which means that we focus only on these two conditions in $Rc$, i.e., that $Can_{Rc_{\alpha}}(a, \alpha, \emptyset, \emptyset)$ holds for all other cases.

It is easy to check that $Rc$ satisfies the generic properties presented in Section 4.2. We can now prove that the centralized implementation meets the $Rc$ requirements.

**Theorem 1** (Consistency and compliance of C-traces). Any valid C-trace is consistent and compliant with $Rc$.

Consistency follows directly from Definition 2 (trace consistency) and Definition 12 (validity).

In order to prove compliance, we consider a valid C-trace $\theta$ and show that it complies with the two conditions of Definition 9. From Definition 4 (compliance):

• For the first condition, we have to show:
  $\forall u_i \in A, \forall d \in A, \theta \vdash Does(d, u_i, update-profile_p(u_i))$
  $\implies \theta \vdash Enables(sn, u_i, update-profile_p(u_i))$

From the definitions of $Does(a, b, \alpha)$ and $Enables(a, b, \alpha)$, this property can be expanded into:

$\forall p \in R, \exists m \in N, \theta_m = S\text{-do-upd-profile}(u_i, p)$
$\implies \exists p \in R, \exists m \in N, \theta_m = S\text{-do-upd-profile}(u_i, p)$
$\implies \exists p \in R, \exists m \in N, \theta_m = S\text{-do-upd-profile}(u_i, p)$
$\implies \exists p \in R, \exists m \in N, \theta_m = S\text{-do-upd-profile}(u_i, p)$
Theorem 2 follows directly from the definitions of relative action control and relative authorization control in Section 3 (Definition 5 and Definition 7 respectively), the definition of Req (Definition 9) and Theorem 1. It expresses the fact that:

- Agents \( u_i \) may consider that they control the updates of their profile only if they trust the social network \( sn \).
- Agents \( u_i \) can forbid connections to their profiles but they are not the only actors with this ability.

5.2 Peer to Peer SNS Implementation

In this section, we consider a fully decentralized implementation of the social network described in Section 4, in which each agent manages their profile on their own node. In contrast with the previous one, this implementation does not involve any dedicated \( sn \) agent.

As was done in the previous section, we first characterize the execution traces of the peer to peer implementation in Section 5.2.1 before defining the trace properties \( Requests(a, a) \), \( Enables(a, b, a) \), and \( Does(a, b, a) \) in Section 5.2.2. Then, we can establish the types and levels of control provided by the peer to peer implementation in Section 5.2.3.

5.2.1 Execution Traces. Concrete traces in the peer to peer implementation are sequences of the following events:

- \( \theta_m = \text{S-do-upd-profile}(u_i, p) \) \( \land \theta \delta(n, m', \text{U-req-upd-profile}(u_i, p)) \)

which is obviously true.

- For the second condition, we have to show:
  \[ \forall u_i \in A, \forall d \in A, \theta \vdash \text{Does}(d, u_i, \text{connect}_a(u_j)) \]

From the definitions (Section 5.1.2) of \( \text{Does}(a, b, a) \) and \( Enables(a, b, a) \) the above property can be expanded into:

\[ \exists m \in \mathbb{N}, \theta_m = \text{S-do-cnn}(u_i, u_j) \land \theta \delta(n, m, \text{U-req-cnn}(u_i, u_j)) \]
\[ \implies \exists m' \in \mathbb{N}, \theta_m = \text{S-transfer-cnn}(u_i, u_j) \land \theta \delta(n, m', \text{U-req-cnn}(u_i, u_j)) \]
\[ \land \exists m'' \in \mathbb{N}, \theta_m = \text{U-accept-cnn}(u_i, u_j) \land \theta \delta(n, m'', \text{U-req-cnn}(u_i, u_j)) \]

This property follows from the third item of Definition 12 (valid C-traces).

Theorem 2 (Control under centralized SNS). The centralized implementation provides:

- Relative action control on update-profile\(_a(u_i)\) to agents \( u_i \) such that \( \text{Trust}(u_i, sn) \).
- No action control on update-profile\(_a(u_i)\) to agents \( u_i \) such that \( \neg \text{Trust}(u_i, sn) \).
- Relative authorization control to agents \( u_i \) on connect\(_a(u_j)\).

We remark that all event names start with "U" as users are assimilated to their node of the social network in this fully decentralized model.

A definition of valid P-traces is given in Appendix A (Definition 13). To follow the paper, it is sufficient to understand that in a valid P-trace a user does not address requests that have not been emitted by another agent.

5.2.2 Trace Properties. In order to establish the control requirements provided by the peer to peer implementation as defined in Section 3, we must first define the trace properties \( Requests(a, a) \), \( Enables(a, b, a) \), and \( Does(a, b, a) \) in terms of execution traces.

- \( \theta \vdash \text{Requests}(u_i, \text{update-profile}_a(u_i)) \)
  \[ \iff \exists p \in \mathcal{R}, \theta_n = \text{U-do-upd-profile}(u_i, p) \]
- \( \theta \vdash \text{Requests}(u_i, \text{access-profile}_a(u_i)) \)
  \[ \iff \exists p \in \mathcal{R}, \theta_n = \text{U-do-acc-profile}(u_i, p) \]
- \( \theta \vdash \text{Requests}(u_i, \text{disconnect}_a(u_j)) \)
  \[ \iff \exists p \in \mathcal{R}, \theta_n = \text{U-do-disconnect}(u_i, p) \]

5.2.3 Control Properties. As was done in Section 5.1.3, we focus on two types of control here:

- Action control of users \( u_i \) over the update of their profiles.
- Authorization control of users \( u_i \) over the connections to their profile (requested by other users).

To express these types of control, we introduce the following requirement.

Definition 10 (Rp Requirement). Requirement \( Rp \) is defined by:

1. \( \text{Can}_{Rp}^{\text{Req}}(u_i, \text{update-profile}_a(u_i)) = \emptyset \)
2. \( \text{Can}_{Rp}^{\text{Req}}(u_i, \text{connect}_a(u_j)) = \{ u_j \} \)

All other sets are considered empty, which means that we focus only on these two conditions in \( Rp \), i.e., that \( \text{Can}^{\text{Req}}(a, a, \emptyset, \emptyset) \) holds for all other cases.

It is easy to check that \( Rp \) satisfies the generic properties presented in Section 4.2. We can now prove that the peer to peer implementation meets the \( Rp \) requirements.

Theorem 3 (Consistency and compliance of P-traces). Any valid P-trace is consistent and compliant with \( Rp \).
Consistency follows directly from Definition 2 (trace consistency) and Definition 13 (validity).
In order to prove compliance, we consider a valid P-trace \( \theta \) and show that it complies with the two conditions of Definition 10.

- The first condition is straightforward (empty set).
- To prove the second condition, we need (Definition 4):
  \[
  \forall u_i \in A, \forall d \in A, \theta \vdash \text{Does}(d, u_i, \text{connect}_u(u_i)) \\
  \implies \theta \vdash \text{Enables}(u_i, u_j, \text{connect}_u(u_j))
  \]

From the definitions (Section 5.1.2) of \( \text{Does}(a, b, a) \) and Enables\((a, b, a)\), the above property can be expanded into:

- \( \exists m \in N, \theta_m = \text{U-accept-req-conn}(u_j, u_i) \land \Omega_{\theta}(n, m, \text{U-req-conn}(u_j, u_i)) \)
- \( \implies \exists m' \in N, \theta_{m'} = \text{U-accept-req-conn}(u_j, u_i) \land \Omega_{\theta}(n, m', \text{U-req-conn}(u_j, u_i)) \)

which is obviously true.

**Theorem 4** (Control under peer to peer SNS). *The peer to peer implementation provides:
- Absolute action control on update-profile\( _n(u_i) \) to agents \( u_i \).
- Absolute authorization control to agents \( u_i \) on connect\( _u(u_i) \).

Theorem 4 follows directly from the definitions of absolute action control and absolute authorization control in Section 3 (Definition 5 and Definition 7 respectively), the definition of \( Rp \) (Definition 10) and Theorem 3. It expresses the fact that:
- Agents \( u_i \) do not depend on others to update their profile.
- Agents \( u_i \) can forbid connections to their profiles and they are the only actors with this ability.

### 5.3 Federated SNS Implementation

In this section, we consider a partially decentralized implementation of the social network system described in Section 4, where each agent potentially shares their node with others, and may or may not trust their node.

Again, we characterize the execution traces of the federated implementation in Section 5.3.1 before defining the trace properties Requests\((a, a)\), Enables\((a, b, a)\), and Does\((a, b, a)\) in Section 5.3.2. Then, we can establish the types and levels of control provided by the federated implementation in Section 5.3.3.

In this implementation, there are a number of nodes running an instance of the SNS software, which we model by adding:

- A number of agents \( s_0, s_1, \ldots \) to \( A \), and
- A Node\((u_i, s_j)\) relation meaning that User\((u_i)\) holds and \( u_i \) uses the social network via the node \( s_j \).

For the sake of simplicity, each user uses only one node, i.e., we have that \( \forall u_i, s_j, s_k \in A, \text{Node}(u_i, s_j) \land \text{Node}(u_i, s_k) \implies s_j = s_k \).

In addition, some users may trust their node (e.g., if they are among the node’s administrator and the implementation they run is open source). In such cases we have \( \text{Node}(u_i, s_j) \land \text{Trust}(u_i, s_j) \).

#### 5.3.1 Execution Traces

Concrete traces in the federated implementation are sequences of the following events:

- \( U-\text{req-upd-profile}(u_i, p, s_j) \): \( u_i \) sends an update \( p \) of their profile \( p_j \) to \( s_j \).
- \( S-\text{do-upd-profile}(s_j, u_i, p) \): \( s_j \) sets \( u_i \)'s profile to \( p \).
- \( U-\text{req-acc-profile}(u_i, u_j, s_k) \): \( u_i \) sends to \( s_k \) a request to access \( u_j \)'s profile.
- \( S-\text{req-profile}(s_k, s_j, u_i, u_j) \): \( s_k \) requests \( u_j \)'s profile to \( s_j \) on behalf of \( u_i \).
- \( S-\text{send-profile}(s_j, u_i, u_j) \): \( s_j \) sends \( u_j \)'s profile to \( s_k \) for \( u_i \).
- \( S-\text{do-acc-profile}(s_k, u_i, u_j) \): \( s_k \) gives access to \( u_j \)'s profile to \( u_i \).
- \( S-\text{req-conn}(u_i, u_j, s_k) \): \( u_i \) sends to \( s_k \) a request to be connected to \( u_j \).
- \( S-\text{req-conn}(s_k, u_i, u_j) \): \( s_k \) forwards to \( s_j \) the connection request from \( u_i \) to \( u_j \).
- \( S-\text{transfer-req-conn}(s_j, u_i, u_j) \): \( s_j \) asks \( u_i \) about the connection request from \( u_i \).
- \( U-\text{accept-req-conn}(u_i, u_j) \): \( u_i \) accepts \( u_j \)'s connection request.
- \( U-\text{reject-req-conn}(u_i, u_j) \): \( u_i \) rejects \( u_j \)'s connection request.
- \( S-\text{do-conn}(s_j, s_k, u_i, u_j) \): \( s_j \) sets up the connection between \( u_i \) and \( u_j \) via \( s_k \).

A definition of valid F-traces is given in Appendix A (Definition 14). To follow the paper, it is sufficient to understand that in a valid F-trace the nodes do not act spontaneously, in particular no action can be performed on behalf of an agent if this agent has not previously requested this action.

#### 5.3.2 Trace Properties

In order to establish the control requirements provided by the federated implementation as defined in Section 3, we must first define the trace properties Requests\((a, a)\), Enables\((a, b, a)\), and Does\((a, b, a)\) in terms of execution traces.

- \( \theta \vdash \text{Requests}(u_i, \text{update-profile}_n(u_i)) \)
  \[\iff \exists s_j \in A, \text{Node}(u_i, s_j), \exists p \in R, \thetap = \text{U-req-upd-profile}(u_i, p, s_j)\]
- \( \theta \vdash \text{Requests}(u_i, \text{access-profile}_n(u_j)) \)
  \[\iff \exists s_k \in A, \text{Node}(u_i, s_k), \thetap = \text{U-req-acc-profile}(u_i, u_j, s_k)\]
- \( \theta \vdash \text{Requests}(u_i, \text{connect}_u(u_j)) \)
  \[\iff \exists s_k \in A, \text{Node}(u_i, s_k), \thetap = \text{U-req-conn}(u_i, u_j, s_k)\]
- \( \theta \vdash \text{Enables}(s_j, u_i, \text{update-profile}_n(u_i)) \)
  \[\iff \text{Node}(u_i, s_j) \land \exists p \in R, \exists m \in N, \thetap = \text{S-send-upd-profile}(u_i, p, s_j) \land \Omega_{\theta}(n, m, \text{U-req-upd-profile}(u_i, p, s_j))\]
- \( \theta \vdash \text{Enables}(s_k, u_i, \text{access-profile}_n(u_j)) \land \text{Node}(u_i, s_k) \)
  \[\iff \exists s_j \in A, \text{Node}(u_i, s_j), \exists m \in N, \thetap = \text{S-send-profile}(u_i, s_j, u_j, s_k) \land \Omega_{\theta}(n, m, \text{U-req-acc-profile}(u_i, u_j, s_k))\]
- \( \theta \vdash \text{Enables}(s_j, u_i, \text{connect}_u(u_j)) \land \text{Node}(u_i, s_j) \)
  \[\iff \exists s_k \in A, \text{Node}(u_i, s_k), \exists m \in N, \thetap = \text{S-send-conn}(s_k, s_j, u_i, u_j) \land \Omega_{\theta}(n, m, \text{U-req-conn}(u_i, u_j, s_k))\]
- \( \theta \vdash \text{Enables}(s_j, u_i, \text{connect}_u(u_j)) \land \text{Node}(u_i, s_j) \)
Thus we introduce the following control requirement.

\[ \exists s_j \in \mathcal{A}, \text{Node}(u_i, s_j), \exists \alpha \in \mathcal{A}, \text{connect}_\alpha(u_i) \]
\[ \theta_m = \text{S-transfer-req-conn}(s_j, u_i) \land \theta_m = \text{U-req-upd-profile}(u_i, u_j, s_j) \land \theta_m = \text{U-req-upd-profile}(u_i, s_j) \land \text{connect}_\alpha(u_i) \]
\[ \theta \vdash \text{enable}(u_i, u_j, \text{connect}_\alpha(u_i)) \]

5.3.2 Control Properties. As for the centralized and peer to peer implementations, we focus on two types of control here:

- Action control of users \( u_i \) over the update of their own profiles.
- Authorization control of users \( u_i \) over the connections to their profile (requested by other users).

Thus we introduce the following control requirement.

**Definition 11 (RF Requirement).** Requirement \( RF \) is defined by:

1. \( \text{Can}_R^F(u_i, \text{update-profile}_i(u_j)) = \{s_j\}, \) where \( \text{Node}(u_i, s_j) \).
2. \( \text{Can}_R^F(u_i, \text{connect}_i(u_j)) = \{s_k, s_j, u_j\}, \) where \( \text{Node}(u_i, s_k) \) and \( \text{Node}(u_j, s_j) \).

All other sets are considered empty, which means that we focus only on these two conditions in \( RF \), i.e., that \( \text{Can}_R(a, \alpha, \emptyset, \emptyset) \) holds for all other cases.

It is easy to check that \( RF \) satisfies the generic properties presented in Section 4.2. We can now prove that the centralized implementation meets the \( RF \) requirements.

**Theorem 5 (Consistency and compliance of F-traces).** Any valid F-trace is consistent and compliant with \( RF \).

Consistency follows directly from Definition 2 (trace consistency) and Definition 14 (validity).

In order to prove compliance, we consider a valid F-trace \( \theta \) and show that it complies with the two conditions of Definition 11. From Definition 4 (compliance):

- For the first condition, we have to show:
  \[ \forall u_i, s_j \in \mathcal{A}, \text{Node}(u_i, s_j), \forall u_i \in \mathcal{A}, \theta \vdash \text{Does}(u_i, u_j, \text{update-profile}_i(u_j)) \]
  \[ \Rightarrow \theta \vdash \text{enable}(s_j, u_i, \text{update-profile}_i(u_j)) \]
  From the definitions (Section 5.3.2) of \( \text{Does}(a, b, a) \) and \( \text{enable}(a, b, a) \), the above property can be expanded into:
  \[ \exists d \in \mathcal{A}, \text{Node}(u_i, d), \exists \alpha \in \mathcal{A}, \exists \alpha \in \mathcal{A}, \text{connect}_\alpha(u_i) \]
  \[ \theta_m = \text{S-do-upd-profile}(d, u_i, p) \land \theta_m = \text{U-req-upd-profile}(u_i, u_j, s_j) \land \theta_m = \text{U-req-upd-profile}(u_i, s_j) \land \text{connect}_\alpha(u_i) \]
  \[ \Rightarrow \exists \beta \in \mathcal{A}, \exists \beta \in \mathcal{A}, \exists \beta \in \mathcal{A}, \text{connect}_\beta(u_i) \]

which is obviously true.

- For the second condition, we have to show:
  \[ \forall u_i, s_j \in \mathcal{A}, \exists u_i \in \mathcal{A}, \theta \vdash \text{Does}(u_i, u_j, \text{connect}_i(u_j)) \]
  \[ \Rightarrow \exists \beta \in \mathcal{A}, \exists \beta \in \mathcal{A}, \exists \beta \in \mathcal{A}, \text{connect}_\beta(u_i) \]
  From the definitions (Section 5.3.2) of \( \text{Does}(a, b, a) \) and \( \text{enable}(a, b, a) \), the above property can be expanded into:
  \[ \exists d \in \mathcal{A}, \text{Node}(u_i, d), \exists \alpha \in \mathcal{A}, \text{Node}(u_i, s_k), \exists \alpha \in \mathcal{A}, \exists \alpha \in \mathcal{A}, \text{connect}_\alpha(u_i) \]
  \[ \theta_m = \text{S-do-upd-profile}(d, s_k, u_i) \land \theta_m = \text{U-req-upd-profile}(u_i, s_j) \land \theta_m = \text{U-req-upd-profile}(u_i, s_j) \land \text{connect}_\alpha(u_i) \]

This property follows from the third item of Definition 14 (valid F-traces).

**Theorem 6 (Control under federated SNS).** The federated implementation provides:

- Relative action control on \( \text{update-profile}_i(u_j) \) to agents \( u_j \) such that \( \text{Node}(u_i, s_j) \land \text{Trust}(u_i, s_j) \).
- No action control on \( \text{update-profile}_i(u_j) \) to agents \( u_j \) such that \( \text{Node}(u_i, s_j) \land \neg \text{Trust}(u_i, s_j) \).
- Relative authorization control to agents \( u_j \) on \( \text{connect}_i(u_j) \).

Theorem 6 follows directly from the definitions of relative action control and relative authorization control in Section 3 (Definition 5 and Definition 7 respectively), the definition of \( RF \) (Definition 11) and Theorem 5. It expresses the fact that:

- Agents \( u_i \) may consider that they control the updates of their own profile only if they trust their node \( s_j \).
- Agents \( u_i \) can forbid connections to their profiles but they are not the only actors with this ability.

5.4 Discussion

We have presented three architectural choices for implementing the social network system introduced in Section 4, namely a centralization model (Section 5.1), a peer to peer implementation (Section 5.2), and a federated implementation (Section 5.3). The Capacity model makes it possible to highlight the different types of control provided by these architectural choices.

The centralized implementation gives users relative control over the update of their profile and connections to them, provided that they trust the social network. This would argue in favor of free and open source software for example, as transparency and auditable code is likely to be more trustworthy for users.
The peer to peer implementation gives users the best control over their personal data: they have absolute control over the update of their profile and over who can connect to them.

The federated implementation is, in terms of control, very close to the centralized implementation. Note however that the assumption \( \text{Trust}(u_i, s_j) \) when Node\((u_i, s_j)\) in the federated case is much more realistic than the assumption \( \text{Trust}(u_i, sn) \) in the centralized case, as nodes may be operated by users themselves or by people they trust (friends, associations, etc.). In practice, the federated implementation is also more likely to be open source, as its developers do not intend to keep users locked-in their own centralized service.

Because the case study used here is very simple, none of these remarks is really surprising. Nevertheless, it confirms that the intuitive notion of control is well captured by Capacity. The added-value of the approach is the fact that these results have been obtained formally through a systematic study of the different implementations. The same approach can be applied to the analysis of more complex and realistic systems. For example, the CNIL\(^{14}\) recently stated\(^{15}\) that biometric access control on smartphones is acceptable because the biometric data processing is performed under the control of the user. It is not clear, however, in what sense users really control their biometric template, what actions they can perform, enable or observe and what actors they have to trust (in addition to the smartphone provider). The same questions hold for many devices in the internet of things.

The fact that the three implementations studied here implement the generic SNS requirement presented in Section 4.2 suggests the potential benefit of defining relation orders on requirements in Capacity. This is further studied in Section B.

Due to lack of space, many aspects of Capacity have not been illustrated in this paper. For example, the only type of context used in our case study is the index of an action in a trace. Contexts can actually be used to express different types of contextual information, in the spirit of contextual integrity\(^{1}\). Other possibilities which have not been illustrated here include personal data of multiple agents, such as pictures involving several friends, and more complex information flows. The first situation can be expressed in a natural way in Capacity as the Pers relation is not exclusive. As far as complex information flows are concerned, we can consider, for instance, a situation where agent \( A \) grants permission to agent \( B \) to access a data \( d \) but prohibits \( C \) to do so. If \( B \) is able to grant access to this data to \( C \), a model of the system in Capacity would show that \( A \) can only have relative control on this data (if they trust \( B \)).

6 RELATED WORK

The two main bodies of work related to the Capacity model presented in this paper concern respectively formal privacy policy languages and usage control models. We sketch successively these two trends of work before discussing the main points of departure of the approach followed in this paper.

Policy languages. Several languages have been proposed for the definition of privacy policies. They differ mostly in terms of scope (general purpose or specific), target (individual privacy policies, corporate rules, legal rules), and semantics. For this kind of tools to be considered as legitimate means to deliver user consent from a legal point of view, they must be able to express unambiguous choices. One of the criticisms raised against early privacy frameworks such as P3P\(^{16}\) was precisely their lack of clarity and the divergent interpretations of privacy policies. An option to solve the ambiguity problem is to resort to a sound, mathematical definition of the semantics of the language. This approach has been followed in several proposals. For example, CI\(^{1}\) is a dedicated linear temporal logic language inspired by the notion of contextual integrity, which makes it possible to express the conditions that have to be met for an agent, acting in a given role and context, to be allowed to transmit a piece of information. CI makes it possible to express both positive and negative norms, and focuses on the transfer of personal data which forms the core of contextual integrity. Another example of temporal logic based privacy policy language is the language proposed in\(^{2}\) which relies on alternating-time temporal logic.

Other languages such as SAP\(^ {3}\) and SIMPL\(^ {10}\) rely on a trace semantics. For example, in SIMPL, users can express their policies using sentences such as “I consent to disclose my CV to a third party only if their privacy policy includes the following commitments: only use this data for the purpose of human resource management; delete this data within a maximum delay of three months; do not transfer this data to any third party”.

Particular attention has been paid to privacy policy languages in the specific context of social networks. These proposals are welcome considering that social networks generally provide many privacy options or parameters whose combinations are difficult to grasp. For example, the models presented in\(^ {6}\) (which generalizes the access control paradigm) and\(^ {12}\) (based on epistemic and deontic properties) can be used not only to better understand the effect of a privacy policy but also to investigate alternative options that are not necessarily supported by existing social networks.

Because they are endowed with a formal semantics in a mathematical framework, the above privacy policy languages make it possible to prove certain properties about the policies (e.g., that a given third party may never receive a given piece of data) and to prove that a given implementation is consistent with the semantics — in other words, that the system behaves as expected by the user.

Access and usage control models. The other most relevant body of literature concerns access and usage control models\(^ {9, 13, 14}\). Many models have been proposed in the computer security area to characterize the conditions under which subjects can access (or use) certain resources. These models include, inter alia, Mandatory Access Control, Discretionary Access Control, and Role-Based Access Control. Usage Control models have been proposed to make it possible to define not only the access rules but also conditions on the use of a resource (e.g., obligations, limitations, etc.). One of the most ambitious frameworks for usage control is the UCON\(ABC\) model, which makes it possible to express authorizations, obligations, and conditions (contextual constraints) at different points of time (before, during, and after usage). Other major features of UCON\(ABC\) are mutable attributes (e.g., actions changing the value

---

\(^{14}\)French data protection authority.

\(^{15}\)https://www.cnil.fr/fr/biometrie-dans-les-smartphones-loi-informatique-et-libertes-exemption-ou-autorisation

\(^{16}\)https://www.w3.org/P3P/
of an attribute) and the continuity of decisions (i.e., rights may be terminated during the usage when attributes have changed). UCON_ABC is very general and can be used to define several families of models. Different approaches have been proposed to define its semantics (or the semantics of a subset of its features), including temporal logic frameworks such as TLA [17] or ITL [4], and process algebra [9].

Capacity. The main point of departure between all the above works and the approach followed in this paper is the fact that the objective of Capacity is not to specify privacy policies or usage control policies, but to define the notion of control itself and to characterize different types of control. For example, UCON_ABC being a usage control framework it focuses on the users of a system rather than on the data subjects as it is necessary to properly express privacy requirements and properties. Moreover, Capacity can be seen as a metamodel with respect to privacy policy or usage control models in the sense that it makes it possible to talk about (express, manipulate) notions which are hardwired in these frameworks. Let us take some examples to illustrate this difference of points of view:

Firstly, neither in privacy policy languages nor in usage control models is it possible to express that an agent depends (or does not depend) on other agents to exercise their rights. Some dependencies could be expressed by temporal properties in privacy policy languages (such as, for example, action \(a\) performed by agent \(a\) cannot occur unless action \(\beta\) performed by agent \(b\) – which could be used to express consent for example – has occurred before). But this would have to be done on a case by case basis, for each relevant action and data (the types of control defined in Section 3, for example, cannot be expressed in these frameworks).

Secondly, both UCON_ABC and privacy policy languages assume the existence of an underlying system to manage the rights. The fact that an agent depends on this system to exercise their rights cannot therefore be expressed within the framework itself. This point was raised as the “administrative issues” in UCON and left for further work [13]. Similarly, the privacy languages mentioned above do not make it possible to refer explicitly to the social network itself (the administrator, in UCON terminology). Therefore it is not possible to express the fact that a user can be more or less dependent on the social network. We believe that this possibility is essential in the context of privacy because data controllers (such as social network providers) can in many cases represent the main source of risk for the user. It is therefore necessary to be able to include them in any model of control. In Capacity, the social network is an agent and it is possible to express the level of dependency of the users with respect to this agent just like their dependencies with respect to other users.

Finally, because Capacity is a metamodel rather than a model, it does not make sense to talk about an architecture to implement it (akin, for example, to reference monitors, as discussed in [9]). Requirements in Capacity define abstract control constraints on systems and these constraints have to be refined to establish a link with an actual implementation. For example, the notion of an agent enabling an action is not limited to a matter of granting rights. Enabling can be implemented in many different ways (e.g., forwarding a message, modifying a data on behalf of the requestor, etc.). Access or usage rights are just particular cases for the implementation of Capacity requirements.

7 CONCLUSION

The main objective of this paper was to introduce the Capacity framework and to show its relevance to provide a formal account of the notion of control. The goal of the social network example developed in the previous sections was only to make the Capacity framework more concrete and to show its use to compare different implementations of a system according to control criteria. We have considered only simple social network functionalities and two aspects of control in this example (action control and authorization control). Other aspects of control corresponding to more complex privacy policies can be represented in the same spirit. Other dimensions of privacy can also be expressed in the Capacity framework. In particular, the purpose for which personal data are processed is very important with regard to privacy, and contexts of actions in Capacity can be used to specify such information. Contexts can also be more complex, including, for example, time, space, or environment (work, family, medical, etc.).

Going one step further, additional developments allowing refinement in the use of the Capacity framework to formally capture the notion of exposure [11] better than using contexts would be beneficial. Indeed, from a legal point of view, for an event to happen in the public space or in a private space depends on the existence of a community of interest (e.g., on a social network, direct contacts of a particular user that this user has manually approved one by one), which can be modeled using relations as defined in Section 2.3. However, from a privacy point of view, this legal definition is not sufficient as for example the size of a community may be as important as its public or private characterization.

The study of non compact requirements, such as what could be implemented with some types of secret sharing techniques [15] (typically with \((t, n)\)-threshold scheme), would also be interesting. A complementary aspect which has not been discussed in this paper is the verification that a given implementation actually meets the validity properties (Definition 12 and 13) and the completeness property (Definition 3), which ensures that a requested action is always performed if all the enabling agents have actually enabled it. This task pertains to traditional code verification techniques [5].

Another perspective would be the use of the classification of control properties presented in this paper to structure the privacy design space and select appropriate strategies according to the objectives in terms of control [7].

An interesting avenue for further research would be to exploit the systematic classification of control properties presented in Section 3 to help data subjects in the definition of their privacy policies. Indeed, provided that they are supported by user-friendly interfaces, the abstract definitions made possible by the Capacity framework could provide a more systematic and intelligible way to grasp user-defined privacy requirements.
REFERENCES


A FORMAL VALIDITY DEFINITIONS

In this appendix, we present the full definitions of valid C-traces, valid P-traces and valid F-traces referred to in Section 5.1.1, Section 5.2.1 and Section 5.3.1 respectively.

Definition 12 (Valid C-trace). A valid trace for the centralized implementation, or valid C-trace, is a sequence of events (as defined in Section 5.1.1) meeting the following properties:

(1) \forall m \in \mathbb{N}, \theta_m = S\text{-}do\text{-}upd\text{-}profile(u_i, p) \implies \exists n \in \mathbb{N}, n < m \land \theta_n = U\text{-}req\text{-}upd\text{-}profile(u_i, u_j)

(2) \forall m \in \mathbb{N}, \theta_m = S\text{-}do\text{-}acc\text{-}profile(u_i, u_j) \implies \exists n \in \mathbb{N}, n < m \land \theta_n = U\text{-}req\text{-}acc\text{-}profile(u_i, u_j)

(3) \forall m, m' \in \mathbb{N}, \exists n, \theta_n = U\text{-}req\text{-}conn(u_i, u_j) \land \theta_{n'} = S\text{-}transfer\text{-}req\text{-}conn(u_i, u_j) \land \theta_{n''} = U\text{-}accept\text{-}req\text{-}conn(u_i, u_j) \land \langle n, m < n'' < m' < m \land \forall k \in \mathbb{N}, n < k < m, \theta_k \neq U\text{-}req\text{-}conn(u_i, u_j) \rangle

Definition 13 (Valid P-trace). A valid trace for the peer to peer implementation, or valid P-trace, is a sequence of events (as defined in Section 5.2.1) meeting the following properties:

(1) \forall m \in \mathbb{N}, \theta_m = U\text{-}do\text{-}acc\text{-}profile(u_i, u_j) \implies \exists n \in \mathbb{N}, n < m \land \theta_n = U\text{-}req\text{-}acc\text{-}profile(u_i, u_j)

(2) \forall m \in \mathbb{N}, \theta_m = U\text{-}accept\text{-}req\text{-}conn(u_i, u_j) \implies \exists n \in \mathbb{N}, n < m \land \theta_n = U\text{-}req\text{-}conn(u_i, u_j)

(3) \forall m \in \mathbb{N}, \theta_m = U\text{-}reject\text{-}req\text{-}conn(u_i, u_j) \implies \exists n \in \mathbb{N}, n < m \land \theta_n = U\text{-}req\text{-}conn(u_i, u_j)

Definition 14 (Valid F-trace). A valid trace for the federated implementation, or valid F-trace, is a sequence of events (as defined in Section 5.3.1) meeting the following properties:

(1) \forall m \in \mathbb{N}, \theta_m = S\text{-}do\text{-}upd\text{-}profile(s_j, u_i, p) \implies \exists n \in \mathbb{N}, n < m \land \theta_n = U\text{-}req\text{-}upd\text{-}profile(s_j, u_i, u_j)

(2) \forall m \in \mathbb{N}, \theta_m = S\text{-}do\text{-}acc\text{-}profile(s_k, u_i, u_j) \implies \exists n \in \mathbb{N}, n < m \land \exists n', n'' \in \mathbb{N}, n'' < n' < m \land \theta_{n'} = S\text{-}req\text{-}profile(s_k, s_j, u_i, u_j) \land \theta_{n''} = S\text{-}send\text{-}profile(s_j, s_k, u_i, u_j) \land \Omega(n, m, U\text{-}req\text{-}acc\text{-}profile(u_i, u_j, s_k))

(3) \forall m \in \mathbb{N}, \theta_m = S\text{-}do\text{-}conn(s_j, u_i, u_j) \implies \exists n \in \mathbb{N}, n < m \land \exists n', n'', m' \in \mathbb{N}, n < n'' < m' < m \land \theta_{n'} = S\text{-}req\text{-}conn(s_j, s_k, u_i, u_j) \land \theta_{n''} = S\text{-}transfer\text{-}req\text{-}conn(s_j, u_i, u_j) \land \theta_{n'} = U\text{-}accept\text{-}req\text{-}conn(u_i, u_j) \land \Omega(n, m, U\text{-}req\text{-}conn(u_i, u_j, s_k))

B RELATION ORDERS ON REQUIREMENTS

In order to make it possible to reason about the level of control provided by a requirement, we introduce several order relations between requirements in Section B.1 and notions of trace compliance with respect to personal data in Section B.2.

B.1 Comparing Requirements

We first define a general order between requirements before introducing specific orders capturing the notion of level of control over personal data.

Definition 15 (R_1 \geq_{R_2} R_2). The general order relation between requirements, noted \geq, is defined as follows: \forall a \in \mathcal{A}, \forall r \in \mathcal{R}, \text{Pers}(r, a) \land \text{Inf}(r, a) \implies \exists \text{Can}_R^a (a, a) \leq \text{Can}_{R_1}^a (a, a) \land \text{Can}_{R_2}^a (a, a) \leq \text{Can}_{R_2}^a (a, a)

Intuitively, R_1 \geq R_2 means that R_1 is more permissive than R_2: it requires less enablers and less witnesses to perform an action. This first order is very generic and it does not deal specifically with personal data. The next definitions allow us to distinguish requirements based on their constraints on personal data.

Definition 16 (R_1 \geq_{w} R_2). The positive preorder relation between requirements, noted \geq_w, is defined as follows: R_1 \geq_{w} R_2 if and only if \forall a \in \mathcal{A}, \forall r \in \mathcal{R}, \text{Pers}(r, a) \land \text{Inf}(r, a) \implies \exists \text{Can}_W^a (a, a) \leq \text{Can}_{R_1}^a (a, a) \land \text{Can}_{R_2}^a (a, a) \leq \text{Can}_{R_2}^a (a, a)
Intuitively, $R_1 \geq ^a_\preceq R_2$ means that $R_1$ is more permissive than $R_2$ for the manipulation of their personal data by agent $a$: it requires less enablers and less witnesses for $a$ to perform an action on their own personal data.

**Definition 17** ($R_1 \geq ^a_\preceq R_2$). The negative preorder relation between requirements, noted $\geq ^a_\preceq$, is defined as follows: $R_1 \geq ^a_\preceq R_2$ if and only if $\forall b \in \mathcal{A}, \forall a \in \Delta, \forall r \in \mathcal{R}$,

- $\text{Pers}(r, a) \land \text{Inf}(r, a) \land a \in \text{Can}^R_{\preceq}(b, a) \implies a \in \text{Can}^R_{\preceq}(b, a)$
- $\text{Pers}(r, a) \land \text{Inf}(r, a) \land a \in \text{Can}^R_{\preceq}(b, a) \implies a \in \text{Can}^R_{\preceq}(b, a)$

Intuitively, $R_1 \geq ^a_\preceq R_2$ means that $R_1$ provides a greater scrutiny than $R_2$ to agent $a$ on the manipulation of their personal data by others: $a$ can enable or witness more actions performed by other agents on $a$’s personal data.

**Definition 18** ($R_1 \geq ^a_\preceq R_2$). The control preorder relation between requirements, noted $\geq ^a_\preceq$, is defined as follows: $R_1 \geq ^a_\preceq R_2$ if and only if $R_1 \geq ^a_\preceq R_2 \land R_1 \geq ^a_\preceq R_2$.

Intuitively, $R_1 \geq ^a_\preceq R_2$ means that $R_1$ provides a greater control than $R_2$ over their personal data to agent $a$: it is more permissive about what $a$ can do with their personal data and provides $a$ a greater scrutiny over what others can do with $a$’s personal data.

The above preorders can be generalized to all agents in a natural way. For example, $\geq$ is defined as follows: $R_1 \geq C R_2$ if and only if $\forall a \in \mathcal{A}, R_1 \geq C R_2$.

**Theorem 7** (Preorders). $\preceq$ is an order relation and $\geq ^a_\preceq, \geq ^a_\preceq, \geq ^a_\preceq, \geq _\preceq$ are preorder relations.

**Proof.** The property of $\preceq$ results from the facts that $\preceq$ is an order relation and the functions $\text{Can}^R_{\preceq}$ and $\text{Can}^R_c$ together define completely $R$. The other relations are only preorders because their definitions involve only subsets of $\Delta$ (actions involving personal data). As usual, it is possible to derive order relations from these preorderings using the quotient sets of the associated equivalence relation.

\[ \square \]

### B.2 Relative Trace Compliance

Let us now introduce notions of trace compliance relative to the personal data of a given agent $a$.

**Definition 19** ($\theta \models ^a_R R$). The positive compliance of a trace $\theta$ with respect to a requirement $R$ and agent $a$ is defined as follows: $\theta \models ^a_R R$ if and only if $\forall a \in \Delta, \forall d \in \mathcal{A}$, $\theta \vdash \text{Does}(d, a, a), \theta \vdash \text{Pers}(r, a) \land \text{Inf}(r, a)$

- $\forall b \in \text{Can}^R_{\preceq}(a, a), \theta \vdash \text{Enables}(b, a, a)$, and
- $\forall b \in \text{Can}^R_{\preceq}(a, a), \exists c \in \mathcal{A}, \theta \vdash \text{Notifies}(c, b, a, a)$.

Intuitively, $\theta \models ^a_R R$ means that in the execution represented by $\theta$, all actions performed by agent $a$ on their personal data have been enabled by the required agents (members of $\text{Can}^R_{\preceq}(a, a)$) and all the necessary witnesses have been notified (members of $\text{Can}^R_{\preceq}(a, a)$).

Positive trace compliance can be generalized to all agents as follows: $\theta \models _\preceq R$ if and only if $\forall a \in \mathcal{A}, \theta \models _\preceq R$.

**Definition 20** ($\theta \models ^a_R R$). The negative compliance of a trace $\theta$ with respect to a requirement $R$ and agent $a$ is defined as follows: $\theta \models ^a_R R$ if and only if $\forall b \in \mathcal{A}, b \neq a, \forall a \in \Delta, \forall d \in \mathcal{A}$, $\theta \vdash \text{Does}(d, b, a) \land \text{Pers}(r, a) \land \text{Inf}(r, a)$

- $a \in \text{Can}^R_{\preceq}(b, a) \implies \theta \vdash \text{Enables}(b, a, a)$, and
- $a \in \text{Can}^R_{\preceq}(b, a) \implies \exists c \in \mathcal{A}, \theta \vdash \text{Notifies}(c, a, b, a)$.

Intuitively, $\theta \models _\preceq R$ means that in the execution represented by $\theta$, any action performed by another agent $b$ on the personal data of $a$ has been enabled by $a$ (if $a$ is a member of $\text{Can}^R_{\preceq}(b, a)$) and notified to $a$ (if $a$ is a member of $\text{Can}^R_{\preceq}(b, a)$).

Negative trace compliance can be generalized to all agents as follows: $\theta \models _\preceq R$ if and only if $\forall a \in \mathcal{A}, \theta \models _\preceq R$.

**Definition 21** ($\theta \models _\preceq R$). The control trace compliance to a requirement with regard to an agent, is defined as follows: $\theta \models _\preceq R$ if and only if $\theta \models ^a_R R \land \theta \models ^a_R R$.

The control trace compliance is defined as the conjunction of positive and negative trace compliances. The intuition is that the control of an agent over their personal data is greater when they have less constraints on their own actions on their data and more powers to prevent actions from other agents on their data. Control trace compliance can be generalized to all agents as follows: $\theta \models _C R$ if and only if $\forall a \in \mathcal{A}, \theta \models _C R$.

**Theorem 8** (Orders and relative trace compliance). For all requirements $R_1, R_2, \theta$, and agent $a$, we have:

1. If $R_1 \geq R_2$ then $\theta \models R_1 \implies \theta \models R_2$
2. If $R_1 \geq _\preceq R_2$ then $\theta \models ^a R_2 \implies \theta \models ^a R_1$
3. If $R_1 \geq _\preceq R_2$ then $\theta \models ^a R_2 \implies \theta \models ^a R_1$
4. If $R_1 \geq _\preceq R_2$ then $\theta \models ^a R_1 \implies \theta \models ^a R_2$
5. If $R_1 \geq _\preceq R_2$ then $\theta \models ^a R_1 \implies \theta \models ^a R_2$
6. If $R_1 \geq _\preceq R_2$ then $\theta \models ^a R_2 \implies \theta \models ^a R_1$ and $\theta \models ^a R_1 \implies \theta \models ^a R_2$
7. If $R_1 \geq _C R_2$ then $\theta \models ^a R_2 \implies \theta \models ^a R_1$ and $\theta \models ^a R_1 \implies \theta \models ^a R_2$

**Proof.** In order to prove the first property, we assume $R_1 \geq R_2$ and $\theta \models R_1$. In order to show $\theta \models R_2$, let us consider $a \in \mathcal{A}, d \in \mathcal{A}, a \in \Delta$ such that $\theta \vdash \text{Does}(d, a, a)$. From $\theta \models R_1$, we have:

- $\forall b \in \text{Can}^R_{\preceq}(a, a), \theta \vdash \text{Enables}(b, a, a)$, and
- $\forall b \in \text{Can}^R_{\preceq}(a, a), \exists c \in \mathcal{A} \text{ s.t. } \theta \vdash \text{Notifies}(c, b, a, a)$.

Then $R_1 \geq R_2$ entails $\text{Can}^R_c(a, a) \subseteq \text{Can}^R_{\preceq}(a, a)$, which allows us to derive:

- $\forall b \in \text{Can}^R_{\preceq}(a, a), \theta \vdash \text{Enables}(b, a, a)$, and
- $\forall b \in \text{Can}^R_{\preceq}(a, a), \exists c \in \mathcal{A} \text{ s.t. } \theta \vdash \text{Notifies}(c, b, a, a)$

and therefore $\theta \models R_2$. The proofs of the other properties are similar with the switch of order for properties 2 and 3 due to the definition of $R_1 \geq _\preceq R_2$. As explained above, more privacy for agents means more possibilities to act on their own data and less possibilities for others. \[ \square \]