Capacity: an Abstract Model of Control over Personal Data

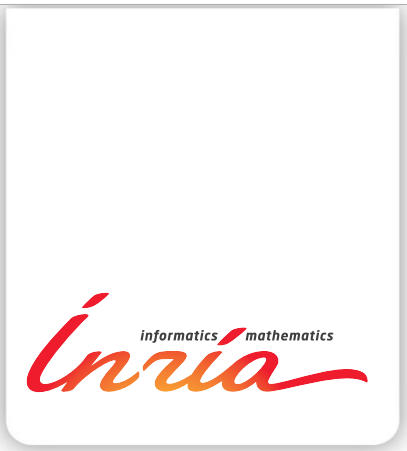
Daniel Le Métayer, Pablo Rauzy

Capacity: an Abstract Model of Control over Personal Data

Daniel Le Métayer, Pablo Rauzy

RESEARCH REPORT
N° 9124
November 2017
Project-Team Privatics

ISSN 0249-6399 ISRN INRIA/RR--9124--FR+ENG



Capacity: an Abstract Model of Control over Personal Data

Daniel Le Métayer, Pablo Rauzy

Project-Team Privatics

Research Report n° 9124 — November 2017 — 21 pages

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.

Key-words: privacy, control, formal model

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Capacity: un modèle abstrait de contrôle sur les données personnelles

Résumé : Tandis que le contrôle des individus sur leurs données personnelles est de plus en plus perçu comme un élément essentiel du respect de leur vie privée, le terme "contrôle" est la plupart du temps utilisé de manière vague, autant par les juristes que les informaticiens. Ce manque de rigueur pourrait causer des mécompréhensions et rend difficile les vérifications de conformité. Pour résoudre ce problème, nous proposons un cadre formel basé sur les capacités pour spécifier la notion de contrôle sur les données personnelles et raisonner sur les propriétés de ce contrôle. Nous appliquons notre cadre formel à des systèmes de réseaux sociaux et montrons qu'il rend possible la caractérisation du type de contrôle des données personnelles que ces systèmes fournissent à leurs utilisateurs ainsi que la comparaison rigoureuse de ces systèmes du point de vu du contrôle.

Mots-clés : vie privée, contrôle, modèle formel

Capacity: an Abstract Model of Control over Personal Data

Daniel Le Métayer[1] and Pablo Rauzy[2]
[1] Inria, Université de Lyon ; daniel.le-metayer@inria.fr
[2] Université Paris 8 / LIASD ; pablo.rauzy@univ-paris8.fr

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.

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[1]. 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).

[1] This principle is often called "informational self-determination" after a ruling of the German Federal Constitutional Court related to the 1983 census.

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 buiding 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 $c_1, c_2, \ldots \in C$, 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 successive 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 in a given context. By convention, we write[2] $\mathsf{o}_c(x_1, \ldots, x_n)$ 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 $\mathsf{o}(x_1, \ldots, x_n)$. Δ denotes the set of actions.

2.3 Relations

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

- $\mathit{Pers}(r, a)$ expresses that resource r is a personal data[3] of agent a,
- $\mathit{In}(r, \alpha)$ means that resource r is involved[4] in action α, and
- $\mathit{Trust}(a, b)$ 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 $\mathit{Can}^R(a, \alpha, E, W) \subseteq \mathcal{A} \times \Delta \times \mathcal{P}(\mathcal{A}) \times \mathcal{P}(\mathcal{A})$ such that $a \notin E$ and $a \notin W$. The intuition for $\mathit{Can}^R(a, \alpha, E, W)$ is that agent a can perform action α 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 α.

By convention, we use $\perp$ to denote an undefined or phantom agent, which never performs nor enables any action. Therefore, $\mathit{Can}^R(a, \alpha, \{\perp\}, W)$ expresses the fact that requirement R prevents a from performing action[6] α. Conversely, $\mathit{Can}^R(a, \alpha, \varnothing, W)$ expresses the fact that requirement R unconditionally allows a to perform action α.

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 $\mathit{Can}^R(a, \alpha, E, W)$:

[2] It should be noted that $\mathsf{o}_c(x_1, \ldots, x_n)$ is not the result of the application of o_c to parameters $x_1, \ldots, x_n$. Formally speaking, it is just a convenient notation for the tuple $[o, c, x_1, \ldots, x_n]$.

[3] A resource can be the personal data of multiple agents. Therefore, we can have $\mathit{Pers}(r, a_1)$ and $\mathit{Pers}(r, a_2)$ with $a_1 \neq a_2$.

[4] More precisely, r is a parameter (or is included in one) of the operation of α.

[5] The notion of trust is intentionally left informal at our level of abstraction.

[6] Specifying that an action is not possible amounts to requiring that this action has to be enabled by an agent that does not exist or never enables any action.

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 $\forall a \in \mathcal{A}$, $\forall \alpha \in \Delta$, $\exists E, W \subseteq \mathcal{A}$, $Can^R(a, \alpha, E, W)$, and $\forall a \in \mathcal{A}$, $\forall \alpha \in \Delta$, $\forall E, E', W, W' \subseteq \mathcal{A}$, $Can^R(a, \alpha, E, W) \wedge Can^R(a, \alpha, E', W') \implies E = E' \wedge W = W'$.

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

- $\overline{Can}_e^R(a, \alpha) = E$ if $\exists W \subseteq \mathcal{A}$ s.t. $Can^R(a, \alpha, E, W)$,
- $\overline{Can}_w^R(a, \alpha) = W$ if $\exists E \subseteq \mathcal{A}$ s.t. $Can^R(a, \alpha, E, W)$,

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

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

1. $\theta \vdash \mathit{Requests}(a, \alpha)$ means that, in trace θ, agent a attempts to perform action α.
2. $\theta \vdash \mathit{Enables}(a, b, \alpha)$ means that, in trace θ, agent a enables the performance of action α by agent b.
3. $\theta \vdash \mathit{Does}(a, b, \alpha)$ means that, in trace θ, agent a performs action α 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. $\theta \vdash \mathit{Notifies}(a, b, c, \alpha)$ means that, in trace θ, agent a notifies to agent b the performance of action α 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.

Definition 2 (Trace consistency)**.** Trace θ is said to be *consistent* if $\theta \vdash \mathit{Does}(c, a, \alpha) \implies \theta \vdash \mathit{Requests}(a, \alpha)$, and $\theta \vdash \mathit{Notifies}(a, b, c, \alpha) \implies \exists d,\ \theta \vdash \mathit{Does}(d, c, \alpha)$

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 θ is said to be *complete* with respect to requirement R if $\theta \vdash \mathit{Requests}(a, \alpha) \wedge \forall b \in \overline{Can}_e^R(a, \alpha),\ \theta \vdash \mathit{Enables}(b, a, \alpha) \implies \exists c,\ \theta \vdash \mathit{Does}(c, a, \alpha)$

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[7].

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 θ is *compliant* with requirement R, noted $\theta \models R$ if and only if $\forall a \in \mathcal{A}$, $\forall d \in \mathcal{A}$, $\forall \alpha \in \Delta$, $\theta \vdash \mathit{Does}(d, a, \alpha) \implies \forall b \in \overline{Can}_e^R(a, \alpha)$, $\theta \vdash \mathit{Enables}(b, a, \alpha)$, and $\forall b \in \overline{Can}_w^R(a, \alpha)$, $\exists c \in \mathcal{A}$, $\theta \vdash \mathit{Notifies}(c, b, a, \alpha)$.

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

[7]For 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.

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 *absolute action control* over action α, noted $AA_R(a, \alpha)$, if and only if $\overline{Can}_e^R(a, \alpha) = \varnothing$.

Requirement R provides agent a *relative action control* over action α, noted $RA_R(a, \alpha)$, if and only if $\forall b \in \mathcal{A}$, $b \in \overline{Can}_e^R(a, \alpha) \implies Trust(a, b)$.

$AA_R(a, \alpha)$ means that a can perform action α without needing any enabler. In other words, a does not depend on any other agent to perform α. In contrast, $RA_R(a, \alpha)$ means that a depends on other agents to perform α but all these agents are trusted by a.

Definition 6 (Observability control)**.** Requirement R provides agent a *absolute observability control* over action α, which is noted $AO_R(a, \alpha)$, if and only if $\overline{Can}_w^R(a, \alpha) = \varnothing$.

Requirement R provides agent a *relative observability control* over action α, which is noted $RO_R(a, \alpha)$, if and only if $\forall b \in \mathcal{A}$, $b \in \overline{Can}_w^R(a, \alpha) \implies Trust(a, b)$.

$AO_R(a, \alpha)$ means that a can perform α discreetly, that is to say without being observable by other agents. In contrast, $RO_R(a, \alpha)$ means that other agents can know that a performs α 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 implicit information flows for example.

Definition 7 (Authorization control)**.** Requirement R provides agent a *absolute authorization control* over action α, which is noted $AH_R(a, \alpha)$, if and only if $\forall b \in \mathcal{A}$ such that $b \neq a$, $\overline{Can}_e^R(b, \alpha) = \{a\}$.

Requirement R provides agent a *relative authorization control* over action α, which is noted $RH_R(a, \alpha)$, if and only if $\forall b \in \mathcal{A}$ such that $b \neq a$, $a \in \overline{Can}_e^R(b, \alpha)$.

$AH_R(a, \alpha)$ means that a can enable the performance of action α by other agents and is the only agent having this power[8]. In other words, the possibility for another agent to perform α depends only on a. In contrast, $RH_R(a, \alpha)$ means that a is not the only agent having this power. In other words, the possibility for another agent to perform α depends not only on a but also on other agents.

Definition 8 (Notification control)**.** Requirement R provides agent a *absolute notification control* over action α, noted $AN_R(a, \alpha)$, if and only if $\forall b \in \mathcal{A}$ such that $b \neq a$, $\overline{Can}_w^R(b, \alpha) = \{a\}$.

Requirement R provides agent a *relative notification control* over action α, noted $RN_R(a, \alpha)$, if and only if $\forall b \in \mathcal{A}$ such that $b \neq a$, $a \in \overline{Can}_w^R(b, \alpha)$.

$AN_R(a, \alpha)$ means that a is informed about the performance of action α by other agents and is the only agent having this power. In contrast, $RN_R(a, \alpha)$ 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 a an *absolute action control* over r, noted $AA_R(a, r)$, iff $\forall \alpha \in \Delta$ s.t. $In(r, \alpha)$, then $AA_R(a, \alpha)$.
- Requirement R provides a an *absolute action control* over their personal data, noted $AA_R(a)$, iff $\forall r \in \mathcal{R}, Pers(r, a) \implies AA_R(a, r)$.

It is easy to check that, for each variant, absolute control implies relative control. Considering that action, observability, authorization, and notification are four independant variants of control, the above definitions give rise to a lattice (using the order defined by implication) made of 81 forms of control[9] (for each action, data, or

[8] We recall that $a \in \overline{Can}_e^R(b, \alpha)$ means that a has the capacity to *prevent* b from performing α.

[9] Which is equal to 3^4 considering that 3 levels are possible for each variant: absolute control, relative control, and lack of control.

agent).

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 all SNS implementations in Section 4.2. 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 $Pers$, $Trust$ and 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 $User$ defined as $User(x) \iff \exists i \in \mathbb{N}, x = u_i$.
- $\mathcal{R}$ 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:
  - $\mathsf{update\text{-}profile}_c(u_i)$ is the update of the profile p_i of user u_i in context c,
  - $\mathsf{access\text{-}profile}_c(u_i)$ is the access to the profile p_i of user u_i in context c,
  - $\mathsf{connect}_c(u_i)$ is the connection[10] 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 *Can*, *Requests*, *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).
- $Pers$ is defined by $Pers(r, a) \iff a = u_i \wedge (r = n_i \vee r = p_i)$, assuming for the sake of simplicity, that agents do not share personal data.
- $Trust$ 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.
- In is derived from the definition of $\mathcal{O}$:

$$\begin{aligned} In(r, \alpha) \iff & (\alpha = \mathsf{update\text{-}profile}_c(u_i) \ \wedge (r = p_i \vee r = n_i)) \\ \vee & \ (\alpha = \mathsf{access\text{-}profile}_c(u_i) \ \wedge (r = p_i \vee r = n_i)) \\ \vee & \ (\alpha = \mathsf{connect}_c(u_i) \wedge r = n_i). \end{aligned}$$

[10] What we mean by connection here is a link in the social network (e.g., following someone on Twitter or adding a friend on Facebook).

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:

$$\begin{array}{l}\forall u_i, u_j \in \mathcal{A}, \mathit{User}(u_i) \wedge \mathit{User}(u_j) \wedge u_i \neq u_j \\ \implies u_j \notin \overline{\mathit{Can}}_e^R(u_i, \mathsf{update\text{-}profile}_c(u_i)) \\ \quad \wedge \quad u_j \notin \overline{\mathit{Can}}_e^R(u_i, \mathsf{access\text{-}profile}_c(u_i)).\end{array}$$

2. A user cannot update the profile of another user:

$$\begin{array}{l}\forall u_i, u_j \in \mathcal{A}, \mathit{User}(u_i) \wedge \mathit{User}(u_j) \wedge u_i \neq u_j \\ \implies \bot \in \overline{\mathit{Can}}_e^R(u_i, \mathsf{update\text{-}profile}_c(u_j)).\end{array}$$

3. A user can always refuse a connection request from another user:

$$\begin{array}{l}\forall u_i, u_j \in \mathcal{A}, \mathit{User}(u_i) \wedge \mathit{User}(u_j) \wedge u_i \neq u_j \\ \implies u_j \in \overline{\mathit{Can}}_e^R(u_i, \mathsf{connect}_c(u_j)).\end{array}$$

4. A user cannot interfere in the action concerning two other users:

$$\begin{array}{l}\forall u_i, u_j, u_k \in \mathcal{A}, u_i \neq u_j \neq u_k \neq u_i, \\ \wedge\ \mathit{User}(u_i) \wedge \mathit{User}(u_j) \wedge \mathit{User}(u_k) \\ \implies u_k \notin \overline{\mathit{Can}}_e^R(u_i, \mathsf{access\text{-}profile}_c(u_j)) \\ \quad \wedge \quad u_k \notin \overline{\mathit{Can}}_e^R(u_i, \mathsf{connect}_c(u_j))\end{array}$$

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 $\mathit{Requests}(a, \alpha)$, $\mathit{Enables}(a, b, \alpha)$, and $\mathit{Does}(a, b, \alpha)$ 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.

- `U-req-upd-profile`(u_i, p): u_i sends an update p of their profile p_i to sn.
- `S-do-upd-profile`(u_i, p): sn sets u_i's profile to p.
- `U-req-acc-profile`(u_i, u_j): u_i sends to sn a request to access u_j's profile.
- `S-do-acc-profile`(u_i, u_j): sn grants u_i's request to access u_j's profile.
- `U-req-conn`(u_i, u_j): u_i sends to sn a request to be connected to u_j.
- `S-transfer-req-conn`(u_i, u_j): sn forwards to u_j the connection request from u_i.

[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 $\mathit{Notifies}(a, b, c, \alpha)$.

- `U-accept-req-conn`(u_i, u_j): u_i accepts u_j's connection request.
- `U-reject-req-conn`(u_i, u_j): u_i rejects u_j's connection request.
- `S-do-conn`(u_i, u_j): sn sets up the connection between u_i and u_j.

In the following, we note θ_n the nth event of an execution trace θ. 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[13] 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 $\mathit{Requests}(a, \alpha)$, $\mathit{Enables}(a, b, \alpha)$, and $\mathit{Does}(a, b, \alpha)$ in terms of execution traces. In the following, the relation Ω_θ is used to relate an event to its trigerring request in trace θ:

$$\Omega_\theta(n, m, \alpha) \iff \theta_n = \alpha \wedge n < m \wedge \forall k, n < k < m \implies \theta_k \neq \alpha.$$

- $\theta \vdash \mathit{Requests}(u_i, \mathsf{update\text{-}profile}_n(u_i))$
  $\iff \exists p \in \mathcal{R}, \theta_n = \texttt{U-req-upd-profile}(u_i, p)$
- $\theta \vdash \mathit{Requests}(u_i, \mathsf{access\text{-}profile}_n(u_j))$
  $\iff \theta_n = \texttt{U-req-acc-profile}(u_i, u_j)$
- $\theta \vdash \mathit{Requests}(u_i, \mathsf{connect}_n(u_j))$
  $\iff \theta_n = \texttt{U-req-conn}(u_i, u_j)$
- $\theta \vdash \mathit{Enables}(sn, u_i, \mathsf{update\text{-}profile}_n(u_i))$
  $\iff \exists p \in \mathcal{R},\ \exists m \in \mathbb{N},$
  $\theta_m = \texttt{S-do-upd-profile}(u_i, p) \wedge$
  $\Omega_\theta(n, m, \texttt{U-req-upd-profile}(u_i, p))$
- $\theta \vdash \mathit{Enables}(sn, u_i, \mathsf{access\text{-}profile}_n(u_j))$
  $\iff \exists m \in \mathbb{N}, \theta_m = \texttt{S-do-acc-profile}(u_i, u_j) \wedge$
  $\Omega_\theta(n, m, \texttt{U-req-acc-profile}(u_i, u_j))$
- $\theta \vdash \mathit{Enables}(sn, u_i, \mathsf{connect}_n(u_j))$
  $\iff \exists m \in \mathbb{N},$
  $\theta_m = \texttt{S-transfer-req-conn}(u_i, u_j) \wedge$
  $\Omega_\theta(n, m, \texttt{U-req-conn}(u_i, u_j))$
- $\theta \vdash \mathit{Enables}(u_i, u_j, \mathsf{connect}_n(u_i))$
  $\iff \exists m \in \mathbb{N}, \theta_m = \texttt{U-accept-req-conn}(u_i, u_j) \wedge$
  $\Omega_\theta(n, m, \texttt{U-req-conn}(u_j, u_i))$
- $\theta \vdash \mathit{Does}(sn, u_i, \mathsf{update\text{-}profile}_n(u_i))$
  $\iff \exists p \in \mathcal{R},\ \exists m \in \mathbb{N},$
  $\theta_m = \texttt{S-do-upd-profile}(u_i, p) \wedge$
  $\Omega_\theta(n, m, \texttt{U-req-upd-profile}(u_i, p))$
- $\theta \vdash \mathit{Does}(sn, u_i, \mathsf{access\text{-}profile}_n(u_j))$
  $\iff \exists m \in \mathbb{N}, \theta_m = \texttt{S-do-acc-profile}(u_i, u_j) \wedge$
  $\Omega_\theta(n, m, \texttt{U-req-acc-profile}(u_i, u_j))$
- $\theta \vdash \mathit{Does}(sn, u_i, \mathsf{connect}_n(u_j))$
  $\iff \exists m \in \mathbb{N}, \theta_m = \texttt{S-do-conn}(u_i, u_j) \wedge$
  $\Omega_\theta(n, m, \texttt{U-req-conn}(u_i, u_j))$

We assume that these conditions define entirely $\mathit{Requests}(a, \alpha)$, $\mathit{Enables}(a, b, \alpha)$, and $\mathit{Does}(a, b, \alpha)$, 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-do-acc-profile`(u_i, u_j) implements both $\mathit{Enables}(sn, u_i, \mathsf{access\text{-}profile}_n(u_j))$ and $\mathit{Does}(sn, u_i, \mathsf{access\text{-}profile}_n(u_j))$ at the same time because granting profile access and providing profile is implemented in a single step by sn in this architecture.

[13] Valid traces for the centralized implementation.

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. $\overline{Can}_e^{Rc}(u_i, \mathsf{update\text{-}profile}_n(u_i)) = \{sn\}$
2. $\overline{Can}_e^{Rc}(u_i, \mathsf{connect}_n(u_j)) = \{sn, u_j\}$

All other sets are considered empty, which means that we focus only on these two conditions in Rc, i.e., that $Can^{Rc}(a, \alpha, \varnothing, \varnothing)$ 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 θ and show that it complies with the two conditions of Definition 9. From Definition 4 (compliance):

- For the first condition, we have to show:

  $$\begin{array}{l}\forall u_i \in \mathcal{A},\ \forall d \in \mathcal{A},\ \theta \vdash \mathit{Does}(d, u_i, \mathsf{update\text{-}profile}_n(u_i)) \\ \implies \theta \vdash \mathit{Enables}(sn, u_i, \mathsf{update\text{-}profile}_n(u_i))\end{array}$$

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

  $$\begin{array}{l}\exists p \in \mathcal{R},\ \exists m \in \mathbb{N},\ \theta_m = \texttt{S-do-upd-profile}(u_i, p) \wedge \\ \Omega_\theta(n, m, \texttt{U-req-upd-profile}(u_i, p)) \\ \implies \exists p \in \mathcal{R},\ \exists m' \in \mathbb{N}, \\ \quad \theta_{m'} = \texttt{S-do-upd-profile}(u_i, p) \wedge \\ \quad \Omega_\theta(n, m', \texttt{U-req-upd-profile}(u_i, p))\end{array}$$

  which is obviously true.

- For the second condition, we have to show:

  $$\begin{array}{l}\forall u_i \in \mathcal{A},\ \forall d \in \mathcal{A},\ \theta \vdash \mathit{Does}(d, u_i, \mathsf{connect}_n(u_j)) \\ \implies \theta \vdash \mathit{Enables}(sn, u_i, \mathsf{connect}_n(u_j)) \wedge \\ \quad \theta \vdash \mathit{Enables}(u_j, u_i, \mathsf{connect}_n(u_j))\end{array}$$

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

  $$\begin{array}{l}\exists m \in \mathbb{N}, \theta_m = \texttt{S-do-conn}(u_i, u_j) \wedge \\ \Omega_\theta(n, m, \texttt{U-req-conn}(u_i, u_j)) \\ \implies \exists m' \in \mathbb{N}, \theta_{m'} = \texttt{S-transfer-req-conn}(u_i, u_j) \\ \quad \wedge\ \Omega_\theta(n, m', \texttt{U-req-conn}(u_i, u_j)) \\ \quad \wedge\ \exists m'' \in \mathbb{N}, \theta_{m''} = \texttt{U-accept-req-conn}(u_j, u_i) \\ \quad \wedge\ \Omega_\theta(n, m'', \texttt{U-req-conn}(u_i, u_j))\end{array}$$

  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* $\mathsf{update\text{-}profile}_n(u_i)$ *to agents* u_i *such that* $\mathit{Trust}(u_i, sn)$*.*
- *No action control on* $\mathsf{update\text{-}profile}_n(u_i)$ *to agents* u_i *such that* $\neg \mathit{Trust}(u_i, sn)$*.*

- *Relative authorization control to agents u_i on* $\mathsf{connect}_n(u_i)$.

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 *Rc* (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 $\mathit{Requests}(a, \alpha)$, $\mathit{Enables}(a, b, \alpha)$, and $\mathit{Does}(a, b, \alpha)$ 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.

- `U-do-upd-profile(`u_i, p`)`: u_i sets their profile to p.
- `U-req-acc-profile(`u_i, u_j`)`: u_i sends to u_j a request to access their profile.
- `U-do-acc-profile(`u_i, u_j`)`: u_i grants u_j's request to access their profile.
- `U-req-conn(`u_i, u_j`)`: u_i sends to u_j a request to be connected to u_j.
- `U-accept-req-conn(`u_i, u_j`)`: u_i accepts u_j's connection request.
- `U-reject-req-conn(`u_i, u_j`)`: u_i rejects u_j's connection request.

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 $\mathit{Requests}(a, \alpha)$, $\mathit{Enables}(a, b, \alpha)$, and $\mathit{Does}(a, b, \alpha)$ in terms of execution traces.

- $\theta \vdash \mathit{Requests}(u_i, \mathsf{update\text{-}profile}_n(u_i))$
  $\iff \exists p \in \mathcal{R}, \theta_n =$ `U-do-upd-profile(`u_i, p`)`
- $\theta \vdash \mathit{Requests}(u_i, \mathsf{access\text{-}profile}_n(u_j))$
  $\iff \theta_n =$ `U-req-acc-profile(`u_i, u_j`)`
- $\theta \vdash \mathit{Requests}(u_i, \mathsf{connect}_n(u_j))$
  $\iff \theta_n =$ `U-req-conn(`u_i, u_j`)`
- $\theta \vdash \mathit{Enables}(u_i, u_j, \mathsf{access\text{-}profile}_n(u_i))$
  $\iff \exists m \in \mathbb{N}, \theta_m =$ `U-do-acc-profile(`u_i, u_j`)`
  $\wedge\ \Omega_\theta(n, m,$ `U-req-acc-profile(`u_j, u_i`))`
- $\theta \vdash \mathit{Enables}(u_i, u_j, \mathsf{connect}_n(u_i))$
  $\iff \exists m \in \mathbb{N}, \theta_m =$ `U-accept-req-conn(`u_i, u_j`)`
  $\wedge\ \Omega_\theta(n, m,$ `U-req-conn(`u_j, u_i`))`

- $\theta \vdash \mathit{Does}(u_i,u_i,\mathsf{update\text{-}profile}_n(u_i))$
  $\iff \exists p \in \mathcal{R}, \theta_n = \texttt{U-do-upd-profile}(u_i, p)$
- $\theta \vdash \mathit{Does}(u_i,u_j,\mathsf{access\text{-}profile}_n(u_i))$
  $\iff \exists m \in \mathbb{N}, \theta_m = \texttt{U-do-acc-profile}(u_i, u_j)$
  $\wedge\ \ \Omega_\theta(n, m, \texttt{U-req-acc-profile}(u_j, u_i))$
- $\theta \vdash \mathit{Does}(u_i,u_j,\mathsf{connect}_n(u_i))$
  $\iff \exists m \in \mathbb{N}, \theta_m = \texttt{U-accept-req-conn}(u_i, u_j)$
  $\wedge\ \ \Omega_\theta(n, m, \texttt{U-req-conn}(u_j, u_i))$

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. $\overline{Can}_e^{Rp}(u_i, \mathsf{update\text{-}profile}_n(u_i)) = \varnothing$
2. $\overline{Can}_e^{Rp}(u_i, \mathsf{connect}_n(u_j)) = \{u_j\}$

All other sets are considered empty, which means that we focus only on these two conditions in *Rp*, i.e., that $Can^{Rp}(a, \alpha, \varnothing, \varnothing)$ 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 θ 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):

  $\forall u_i \in \mathcal{A}, \forall d \in \mathcal{A}, \theta \vdash \mathit{Does}(d, u_i, \mathsf{connect}_n(u_j))$
  $\implies \theta \vdash \mathit{Enables}(u_j, u_i, \mathsf{connect}_n(u_j))$

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

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

  which is obviously true.

Theorem 4 (Control under peer to peer SNS)**.** *The peer to peer implementation provides:*

- *Absolute action control on* $\mathsf{update\text{-}profile}_n(u_i)$ *to agents* u_i*.*
- *Absolute authorization control to agents* u_i *on* $\mathsf{connect}_n(u_i)$*.*

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 $\mathit{Requests}(a, \alpha)$, $\mathit{Enables}(a, b, \alpha)$, and $\mathit{Does}(a, b, \alpha)$ 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 $\mathcal{A}$, and
- a $\mathit{Node}(u_i, s_j)$ relation meaning that $\mathit{User}(u_i)$ 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 $\forall u_i, s_j, s_k \in \mathcal{A}, \mathit{Node}(u_i, s_j) \wedge \mathit{Node}(u_i, s_k) \implies s_j = s_k$. 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 $\mathit{Node}(u_i, s_j) \wedge \mathit{Trust}(u_i, s_j)$.

5.3.1 Execution Traces

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

- `U-req-upd-profile`(u_i, p, s_j): u_i sends an update p of their profile p_i to s_j.
- `S-do-upd-profile`(s_j, u_i, p): s_j sets u_i's profile to p.
- `U-req-acc-profile`(u_i, u_j, s_k): u_i sends to s_k a request to access u_j's profile.
- `S-req-profile`(s_k, s_l, u_i, u_j): s_k requests u_j's profile to s_l on behalf of u_i.
- `S-send-profile`(s_l, s_k, u_j, u_i): s_l sends u_j's profile to s_k for u_i.
- `S-do-acc-profile`(s_k, u_i, u_j): s_k gives access to u_j's profile to u_i.
- `U-req-conn`(u_i, u_j, s_k): u_i sends to s_k a request to be connected to u_j.
- `S-req-conn`(s_k, s_l, u_i, u_j): s_k forwards to s_l the connection request from u_i to u_j.
- `S-transfer-req-conn`(s_l, u_j, u_i): s_l asks u_j about the connection request from u_i.
- `U-accept-req-conn`(u_i, u_j): u_i accepts u_j's connection request.
- `U-reject-req-conn`(u_i, u_j): u_i rejects u_j's connection request.
- `S-do-conn`(s_l, s_k, u_i, u_j): s_l 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 $\mathit{Requests}(a, \alpha)$, $\mathit{Enables}(a, b, \alpha)$, and $\mathit{Does}(a, b, \alpha)$ in terms of execution traces.

- $\theta \vdash \mathit{Requests}(u_i, \mathsf{update\text{-}profile}_n(u_i))$
  $\iff \exists s_j \in \mathcal{A}, \mathit{Node}(u_i, s_j), \exists p \in \mathcal{R},$
  $\theta_n =$ `U-req-upd-profile`(u_i, p, s_j)
- $\theta \vdash \mathit{Requests}(u_i, \mathsf{access\text{-}profile}_n(u_j))$
  $\iff \exists s_k \in \mathcal{A}, \mathit{Node}(u_i, s_k),$
  $\theta_n =$ `U-req-acc-profile`(u_i, u_j, s_k)
- $\theta \vdash \mathit{Requests}(u_i, \mathsf{connect}_n(u_j))$
  $\iff \exists s_k \in \mathcal{A}, \mathit{Node}(u_i, s_k),$
  $\theta_n =$ `U-req-conn`(u_i, u_j, s_k)

- $\theta \vdash \mathit{Enables}(s_j,u_i,\mathsf{update\text{-}profile}_n(u_i))$
  $\iff Node(u_i,s_j) \wedge \exists p \in \mathcal{R}, \exists m \in \mathbb{N},$
  $\theta_m = \texttt{S-do-upd-profile}(s_j,u_i,p) \wedge$
  $\Omega_\theta(n,m,\texttt{U-req-upd-profile}(u_i,p,s_j))$
- $\theta \vdash \mathit{Enables}(s_k,u_i,\mathsf{access\text{-}profile}_n(u_j)) \wedge Node(u_i,s_k)$
  $\iff \exists s_l \in \mathcal{A}, Node(u_j,s_l), \exists m \in \mathbb{N},$
  $\theta_m = \texttt{S-req-profile}(s_k,s_l,u_i,u_j) \wedge$
  $\Omega_\theta(n,m,\texttt{U-req-acc-profile}(u_i,u_j,s_k))$
- $\theta \vdash \mathit{Enables}(s_l,u_i,\mathsf{access\text{-}profile}_n(u_j)) \wedge Node(u_j,s_j)$
  $\iff \exists s_k \in \mathcal{A}, Node(u_i,s_k), \exists m \in \mathbb{N},$
  $\theta_m = \texttt{S-send-profile}(s_l,s_k,u_j,u_i) \wedge$
  $\Omega_\theta(n,m,\texttt{U-req-acc-profile}(u_i,u_j,s_k))$
- $\theta \vdash \mathit{Enables}(s_k,u_i,\mathsf{connect}_n(u_j)) \wedge Node(u_i,s_k)$
  $\iff \exists s_l \in \mathcal{A}, Node(u_j,s_l), \exists m \in \mathbb{N},$
  $\theta_m = \texttt{S-req-conn}(s_k,s_l,u_i,u_j) \wedge$
  $\Omega_\theta(n,m,\texttt{U-req-conn}(u_i,u_j,s_k))$
- $\theta \vdash \mathit{Enables}(s_l,u_i,\mathsf{connect}_n(u_j)) \wedge Node(u_j,s_l)$
  $\iff \exists s_k \in \mathcal{A}, Node(u_i,s_k), \exists m \in \mathbb{N},$
  $\theta_m = \texttt{S-transfer-req-conn}(s_l,u_j,u_i) \wedge$
  $\Omega_\theta(n,m,\texttt{U-req-conn}(u_i,u_j,s_k))$
- $\theta \vdash \mathit{Enables}(u_i,u_j,\mathsf{connect}_n(u_i))$
  $\iff \exists s_k \in \mathcal{A}, Node(u_i,s_k), \exists m \in \mathbb{N},$
  $\theta_m = \texttt{U-accept-req-conn}(u_i,u_j) \wedge$
  $\Omega_\theta(n,m,\texttt{U-req-conn}(u_j,u_i,s_k))$
- $\theta \vdash \mathit{Does}(s_j,u_i,\mathsf{update\text{-}profile}_n(u_i))$
  $\iff Node(u_i,s_j), \exists p \in \mathcal{R}, \exists m \in \mathbb{N},$
  $\theta_m = \texttt{S-do-upd-profile}(s_j,u_i,p) \wedge$
  $\Omega_\theta(n,m,\texttt{U-req-upd-profile}(u_i,p,s_j))$
- $\theta \vdash \mathit{Does}(s_k,u_i,\mathsf{access\text{-}profile}_n(u_j))$
  $\iff Node(u_i,s_k), \exists m \in \mathbb{N},$
  $\theta_m = \texttt{S-do-acc-profile}(s_k,u_i,u_j) \wedge$
  $\Omega_\theta(n,m,\texttt{U-req-acc-profile}(u_i,u_j,s_k))$
- $\theta \vdash \mathit{Does}(s_l,u_i,\mathsf{connect}_n(u_j))$
  $\iff Node(u_j,s_l), \exists s_k \in \mathcal{A}, Node(u_i,s_k), \exists m \in \mathbb{N},$
  $\theta_m = \texttt{S-do-conn}(s_l,s_k,u_i,u_j) \wedge$
  $\Omega_\theta(n,m,\texttt{U-req-conn}(u_i,u_j,s_k))$

5.3.3 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. $\overline{Can}_e^{Rf}(u_i, \mathsf{update\text{-}profile}_n(u_i)) = \{s_j\}$,
where $Node(u_i,s_j)$,

2. $\overline{Can}_e^{Rf}(u_i, \mathsf{connect}_n(u_j)) = \{s_k,s_l,u_j\}$,
where $Node(u_i,s_k)$ and $Node(u_j,s_l)$

All other sets are considered empty, which means that we focus only on these two conditions in Rf, i.e., that $Can^{Rf}(a,\alpha,\varnothing,\varnothing)$ 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 θ and show that it complies with the two conditions of Definition 11. From Definition 4 (compliance):

- For the first condition, we have to show:

$$\begin{array}{l}\forall u_i, s_j \in \mathcal{A}, \mathit{Node}(u_i, s_j),\ \forall d \in \mathcal{A},\\ \theta \vdash \mathit{Does}(d, u_i, \mathsf{update\text{-}profile}_n(u_i))\\ \implies \theta \vdash \mathit{Enables}(s_j, u_i, \mathsf{update\text{-}profile}_n(u_i))\end{array}$$

  From the definitions (Section 5.3.2) of $\mathit{Does}(a, b, \alpha)$ and $\mathit{Enables}(a, b, \alpha)$, the above property can be expanded into:

$$\begin{array}{l}\exists d \in \mathcal{A}, \mathit{Node}(u_i, d),\ \exists p \in \mathcal{R},\ \exists m \in \mathbb{N},\\ \theta_m = \texttt{S-do-upd-profile}(d, u_i, p) \wedge\\ \Omega_\theta(n, m, \texttt{U-req-upd-profile}(u_i, p, d))\\ \implies \exists p \in \mathcal{R},\ \exists m' \in \mathbb{N},\\ \qquad \theta_{m'} = \texttt{S-do-upd-profile}(s_j, u_i, p) \wedge\\ \qquad \Omega_\theta(n, m', \texttt{U-req-upd-profile}(u_i, p, s_j))\end{array}$$

  which is obviously true.

- For the second condition, we have to show:

$$\begin{array}{l}\forall u_i, u_j, s_k, s_l \in \mathcal{A},\ \forall d \in \mathcal{A},\\ \theta \vdash \mathit{Does}(d, u_i, \mathsf{connect}_n(u_j))\\ \implies \theta \vdash \mathit{Enables}(s_k, u_i, \mathsf{connect}_n(u_j)) \wedge\\ \qquad \theta \vdash \mathit{Enables}(s_l, u_i, \mathsf{connect}_n(u_j)) \wedge\\ \qquad \theta \vdash \mathit{Enables}(u_j, u_i, \mathsf{connect}_n(u_j))\end{array}$$

  From the definitions (Section 5.3.2) of $\mathit{Does}(a, b, \alpha)$ and $\mathit{Enables}(a, b, \alpha)$, the above property can be expanded into:

$$\begin{array}{l}\exists d \in \mathcal{A}, \mathit{Node}(u_j, d),\ \exists s_k \in \mathcal{A}, \mathit{Node}(u_i, s_k),\ \exists m \in \mathbb{N},\\ \theta_m = \texttt{S-do-conn}(d, s_k, u_i, u_j) \wedge\\ \Omega_\theta(n, m, \texttt{U-req-conn}(u_i, u_j, s_k))\\ \implies \exists s_l \in \mathcal{A}, \mathit{Node}(u_j, s_l),\ \exists m \in \mathbb{N},\\ \qquad \theta_m = \texttt{S-req-conn}(s_k, s_l, u_i, u_j) \wedge\\ \qquad \Omega_\theta(n, m, \texttt{U-req-conn}(u_i, u_j, s_k))\\ \quad \wedge\ \exists m \in \mathbb{N},\\ \qquad \theta_m = \texttt{S-transfer-req-conn}(s_l, u_j, u_i) \wedge\\ \qquad \Omega_\theta(n, m, \texttt{U-req-conn}(u_i, u_j, s_k))\\ \quad \wedge\ \exists m \in \mathbb{N}, \theta_m = \texttt{U-accept-req-conn}(u_i, u_j) \wedge\\ \qquad \Omega_\theta(n, m, \texttt{U-req-conn}(u_j, u_i, s_k))\end{array}$$

  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* $\mathsf{update\text{-}profile}_n(u_i)$ *to agents* u_i *such that* $\mathit{Node}(u_i, s_j) \wedge \mathit{Trust}(u_i, s_j)$.
- *No action control on* $\mathsf{update\text{-}profile}_n(u_i)$ *to agents* u_i *such that* $\mathit{Node}(u_i, s_j) \wedge \neg \mathit{Trust}(u_i, s_j)$.
- *Relative authorization control to agents* u_i *on* $\mathsf{connect}_n(u_i)$.

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 centralized implementation (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 over 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 $\mathit{Trust}(u_i, s_j)$ when $\mathit{Node}(u_i, s_j)$ in the federated case is much more realistic than the assumption $\mathit{Trust}(u_i, sn)$ 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 S4P [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

[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/`.

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 [13, 9, 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 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 α performed by agent a cannot occur unless action β 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[17]. 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,

[17] As an illustration, according to its privacy policy (as of 2017), Facebook retains the right to use the personal data of its users to deliver ads and measure their effectiveness, to provide location-based services, to make suggestions and to provide innovative features and services it develops in the future. In addition, Facebook may change its terms, and the continued use of the service following changes to the terms constitutes acceptance of the amended terms.

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 refinements 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

[1] Adam Barth, Anupam Datta, John C. Mitchell, and Helen Nissenbaum. Privacy and Contextual Integrity: Framework and Applications. In *IEEE Symposium on Security and Privacy*, 2006.

[2] Adam Barth, Anupam Datta, John C. Mitchell, and Sharada Sundaram. Privacy and Utility in Business Processes. In *20th IEEE Computer Security Foundations Symposium (CSF07)*, 2007.

[3] Moritz Y. Becker, Alexander Malkis, and Laurent Bussard. S4P: A Generic Language for Specifying Privacy Preferences and Policies, 2010.

[4] Antonio Cau and Hussein Zedan. Refining Interval Temporal Logic specifications. In *Transformation-Based Reactive Systems Development*, 1997.

[5] Vijay D'Silva, Daniel Kroening, and Georg Weissenbacher. A Survey of Automated Techniques for Formal Software Verification. In *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2008.

[6] Philip W.L. Fong, Mohd Anwar, and Zhen Zhao. A Privacy Preservation Model for Facebook-Style Social Network Systems. In *ESORICS*, 2009.

[7] Jaap-Henk Hoepman. Privacy Design Strategies. In *ICT Systems Security and Privacy Protection*, 2014.

[8] Christophe Lazaro and Daniel Le Métayer. Control over Personal Data: True Remedy or Fairy Tale? *SCRIPTed*, 2015.

[9] Aliaksandr Lazouski, Fabio Martinelli, and Paolo Mori. Usage control in computer security: A survey. *Computer Science Review 4 (2)*, 2010.

[10] Daniel Le Métayer. A formal privacy management framework. In *Formal Aspects in Security and Trust (FAST 2008)*, 2008.

[11] Mainack Mondal, Peter Druschel, Krishna P. Gummadi, and Alan Mislove. Beyond Access Control: Managing Online Privacy via Exposure. In *Workshop on Useable Security*, 2014.

[12] Raul Pardo and Gerardo Schneider. A Formal Privacy Policy Framework for Social Networks. In *12th International Conference on Software Engineering and Formal Methods (SEFM)*, 2014.

[13] Jaehong Park and Ravi Sandhu. The $UCON_{ABC}$ Usage Control Model. *ACM Transactions on Information and System Security*, 2004.

[14] Jaehong Park and Ravi Sandhu. A Position Paper: A Usage Control (UCON) Model for Social Networks Privacy, 2010.

[15] Adi Shamir. How to share a secret. *Communications of the ACM*, 2008.

[16] Wikipedia. The Right to Privacy. Accessed on 2015-11-24.

[17] Xinwen Zhang, Francesco Parisi-Presicce, Ravi Sandhu, and Jaehong Park. Formal Model and Policy Specification of Usage Control. *ACM Transactions on Information and System Security*, 2005.

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:

$$\begin{aligned}
&\forall m \in \mathbb{N}, \theta_m = \texttt{S-do-upd-profile}(u_i, p) \\
&\implies \exists n \in \mathbb{N}, n < m \wedge \\
&\quad \theta_n = \texttt{U-req-upd-profile}(u_i, p)
\end{aligned}$$

$$\begin{aligned}
&\forall m \in \mathbb{N}, \theta_m = \texttt{S-do-acc-profile}(u_i, u_j) \\
&\implies \exists n \in \mathbb{N}, n < m \wedge \\
&\quad \theta_n = \texttt{U-req-acc-profile}(u_i, u_j)
\end{aligned}$$

$$\begin{aligned}
&\forall m \in \mathbb{N}, \theta_m = \texttt{S-do-conn}(u_i, u_j) \\
&\implies \exists n, m', m'' \in \mathbb{N}, \theta_n = \texttt{U-req-conn}(u_i, u_j) \wedge \\
&\quad \theta_{m'} = \texttt{S-transfer-req-conn}(u_i, u_j) \wedge \\
&\quad \theta_{m''} = \texttt{U-accept-req-conn}(u_j, u_i) \wedge \\
&\quad n < m' < m'' < m \wedge \\
&\quad \forall k \in \mathbb{N}, n < k < m, \theta_k \neq \texttt{U-req-conn}(u_i, u_j)
\end{aligned}$$

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:

$$\begin{aligned}
&\forall m \in \mathbb{N}, \theta_m = \texttt{U-do-acc-profile}(u_i, u_j) \\
&\implies \exists n \in \mathbb{N}, n < m \wedge \\
&\quad \theta_n = \texttt{U-req-acc-profile}(u_j, u_i)
\end{aligned}$$

$$\begin{aligned}
&\forall m \in \mathbb{N}, \theta_m = \texttt{U-accept-req-conn}(u_i, u_j) \\
&\implies \exists n \in \mathbb{N}, n < m \wedge \theta_n = \texttt{U-req-conn}(u_j, u_i)
\end{aligned}$$

$$\begin{aligned}
&\forall m \in \mathbb{N}, \theta_m = \texttt{U-reject-req-conn}(u_i, u_j) \\
&\implies \exists n \in \mathbb{N}, n < m \wedge \theta_n = \texttt{U-req-conn}(u_j, u_i)
\end{aligned}$$

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:

$$\begin{aligned}
&\forall m \in \mathbb{N}, \theta_m = \texttt{S-do-upd-profile}(s_j, u_i, p) \\
&\implies \mathit{Node}(u_i, s_j) \wedge \exists n \in \mathbb{N}, n < m \wedge \\
&\quad \theta_n = \texttt{U-req-upd-profile}(u_i, p, s_j)
\end{aligned}$$

$\forall m \in \mathbb{N}, \theta_m = \texttt{S-do-acc-profile}(s_k, u_i, u_j)$
$\implies Node(u_i, s_k) \wedge Node(u_j, s_l) \wedge$
$\exists n', n'' \in \mathbb{N}, n' < n'' < m \wedge$
$\theta_{n'} = \texttt{S-req-profile}(s_k, s_l, u_i, u_j) \wedge$
$\theta_{n''} = \texttt{S-send-profile}(s_l, s_k, u_j, u_i) \wedge$
$\Omega_\theta(n, m, \texttt{U-req-acc-profile}(u_i, u_j, s_k))$

$\forall m \in \mathbb{N}, \theta_m = \texttt{S-do-conn}(s_l, s_k, u_i, u_j)$
$\implies Node(u_i, s_k) \wedge Node(u_j, s_l) \wedge$
$\exists n', n'', m' \in \mathbb{N}, n' < n'' < m' < m \wedge$
$\theta_{n'} = \texttt{S-req-conn}(s_k, s_l, u_i, u_j) \wedge$
$\theta_{n''} = \texttt{S-transfer-req-conn}(s_l, u_j, u_i) \wedge$
$\theta_{m'} = \texttt{U-accept-req-conn}(u_j, u_i) \wedge$
$\Omega_\theta(n, m, \texttt{U-req-conn}(u_i, u_j, s_k))$

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 ($R_1 \succeq R_2$)**.** The *general order* relation between requirements, noted $\succeq$, is defined as follows: $R_1 \succeq R_2$ if and only if $\forall a \in \mathcal{A}$, $\forall \alpha \in \Delta$,

$\overline{Can}_e^{R_1}(a, \alpha) \subseteq \overline{Can}_e^{R_2}(a, \alpha) \wedge \overline{Can}_w^{R_1}(a, \alpha) \subseteq \overline{Can}_w^{R_2}(a, \alpha)$

Intuitively, $R_1 \succeq R_2$ 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 contraints on personal data.

Definition 16 ($R_1 \succeq^a_\oplus R_2$)**.** The *positive preorder* relation between requirements, noted $\succeq^a_\oplus$, is defined as follows: $R_1 \succeq^a_\oplus R_2$ if and only if $\forall \alpha \in \Delta$, $\forall r \in \mathcal{R}$, $Pers(r, a) \wedge In(r, \alpha) \implies$
$\overline{Can}_e^{R_1}(a, \alpha) \subseteq \overline{Can}_e^{R_2}(a, \alpha) \wedge \overline{Can}_w^{R_1}(a, \alpha) \subseteq \overline{Can}_w^{R_2}(a, \alpha)$

Intuitively, $R_1 \succeq^a_\oplus R_2$ 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 ($R_1 \succeq^a_\ominus R_2$)**.** The *negative preorder* relation between requirements, noted $\succeq^a_\ominus$, is defined as follows: $R_1 \succeq^a_\ominus R_2$ if and only if $\forall b \in \mathcal{A}$, $\forall \alpha \in \Delta$, $\forall r \in \mathcal{R}$,

- $Pers(r, a) \wedge In(r, \alpha) \wedge a \in \overline{Can}_e^{R_2}(b, \alpha) \implies a \in \overline{Can}_e^{R_1}(b, \alpha)$
- $Pers(r, a) \wedge In(r, \alpha) \wedge a \in \overline{Can}_w^{R_2}(b, \alpha) \implies a \in \overline{Can}_w^{R_1}(b, \alpha)$

Intuitively, $R_1 \succeq^a_\ominus R_2$ 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 ($R_1 \succeq^a_C R_2$)**.** The *control preorder* relation between requirements, noted $\succeq^a_C$, is defined as follows: $R_1 \succeq^a_C R_2$ if and only if $R_1 \succeq^a_\oplus R_2 \wedge R_1 \succeq^a_\ominus R_2$.

Intuitively, $R_1 \succeq^a_C R_2$ 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, $\succeq_C$ is defined as follows: $R_1 \succeq_C R_2$ if and only if $\forall a \in \mathcal{A}$, $R_1 \succeq^a_C R_2$.

Theorem 7 (Preorders)**.** *$\succeq$ is an order relation and $\succeq^a_\oplus$, $\succeq^a_\ominus$, $\succeq^a_C$, $\succeq_\oplus$, $\succeq_\ominus$, $\succeq_C$ are preorder relations.*

Proof. The property of $\succeq$ results from the facts that $\subseteq$ is an order relation and the functions $\overline{Can}_w^R$ and $\overline{Can}_e^R$ together define completely R. The other relations are only preorders because their definitions involve only subsets of Δ (actions involving personal data). As usual, it is possible to derive order relations from these preorders using the quotient sets of the associated equivalence relation. □

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 ($\theta \models^a_\oplus R$)**.** The *positive compliance* of a trace θ with respect to a requirement R and agent a is defined as follows: $\theta \models^a_\oplus R$ if and only if $\forall \alpha \in \Delta$, $\forall d \in \mathcal{A}$,
$\theta \vdash \mathit{Does}(d, a, \alpha) \wedge \mathit{Pers}(r, a) \wedge \mathit{In}(r, \alpha) \implies$

- $\forall b \in \overline{\mathit{Can}}^R_e(a, \alpha)$, $\theta \vdash \mathit{Enables}(b, a, \alpha)$, and
- $\forall b \in \overline{\mathit{Can}}^R_w(a, \alpha)$, $\exists c \in \mathcal{A}$, $\theta \vdash \mathit{Notifies}(c, b, a, \alpha)$.

Intuitively, $\theta \models^a_\oplus R$ means that in the execution represented by θ, all actions performed by agent a on their personal data have been enabled by the required agents (members of $\overline{\mathit{Can}}^R_e(a, \alpha)$) and all the necessary witnesses have been notified (members of $\overline{\mathit{Can}}^R_w(a, \alpha)$).

Positive trace compliance can be generalized to all agentsas follows: $\theta \models_\oplus R$ if and only if $\forall a \in \mathcal{A}$, $\theta \models^a_\oplus R$.

Definition 20 ($\theta \models^a_\ominus R$)**.** The *negative compliance* of a trace θ with respect to a requirement R and agent a is defined as follows: $\theta \models^a_\ominus R$ if and only if $\forall b \in \mathcal{A}, b \neq a$, $\forall \alpha \in \Delta$, $\forall d \in \mathcal{A}$,
$\theta \vdash \mathit{Does}(d, b, \alpha) \wedge \mathit{Pers}(r, a) \wedge \mathit{In}(r, \alpha) \implies$

- $a \in \overline{\mathit{Can}}^R_e(b, \alpha) \implies \theta \vdash \mathit{Enables}(a, b, \alpha)$, and
- $a \in \overline{\mathit{Can}}^R_w(b, \alpha) \implies \exists c \in \mathcal{A}$, $\theta \vdash \mathit{Notifies}(c, a, b, \alpha)$.

Intuitively, $\theta \models^a_\ominus R$ means that in the execution represented by θ, any action performed by another agent b on the personal data of a has been enabled by a (if a is a member of $\overline{\mathit{Can}}^R_e(b, \alpha)$) and notified to a (if a is a member of $\overline{\mathit{Can}}^R_w(b, \alpha)$).

Negative trace compliance can be generalized to all agentsas follows: $\theta \models_\ominus R$ if and only if $\forall a \in \mathcal{A}$, $\theta \models^a_\ominus R$.

Definition 21 ($\theta \models^a_C R$)**.** The *control trace compliance* to a requirement with regard to an agent, is defined as follows: $\theta \models^a_C R$ if and only if $\theta \models^a_\oplus R \wedge \theta \models^a_\ominus R$.

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 agentsas follows: $\theta \models_C R$ if and only if $\forall a \in \mathcal{A}$, $\theta \models^a_C R$.

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

1. *If $R_1 \succeq R_2$ then $\theta \models R_1 \implies \theta \models R_2$*
2. *If $R_1 \succeq^a_\oplus R_2$ then $\theta \models^a_\oplus R_2 \implies \theta \models^a_\oplus R_1$*
3. *If $R_1 \succeq_\oplus R_2$ then $\theta \models_\oplus R_2 \implies \theta \models_\oplus R_1$*
4. *If $R_1 \succeq^a_\ominus R_2$ then $\theta \models^a_\ominus R_1 \implies \theta \models^a_\ominus R_2$*
5. *If $R_1 \succeq_\ominus R_2$ then $\theta \models_\ominus R_1 \implies \theta \models_\ominus R_2$*
6. *If $R_1 \succeq^a_C R_2$ then $\theta \models^a_\oplus R_2 \implies \theta \models^a_\oplus R_1$ and $\theta \models^a_\ominus R_1 \implies \theta \models^a_\ominus R_2$*
7. *If $R_1 \succeq_C R_2$ then $\theta \models_\oplus R_2 \implies \theta \models_\oplus R_1$ and $\theta \models_\ominus R_1 \implies \theta \models_\ominus R_2$*

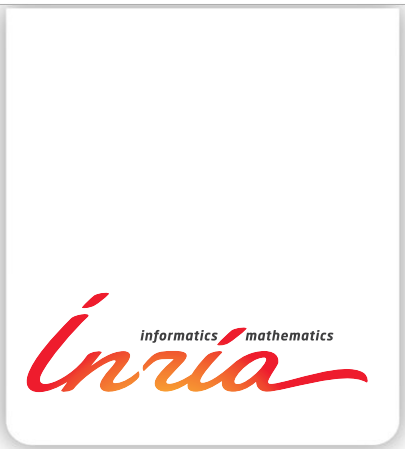
Proof. In order to prove the first property, we assume $R_1 \succeq R_2$ and $\theta \models R_1$. In order to show $\theta \models R_2$, let us consider $a \in \mathcal{A}$, $d \in \mathcal{A}$, $\alpha \in \Delta$ such that $\theta \vdash \mathit{Does}(d, a, \alpha)$. From $\theta \models R_1$, we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

- $\forall b \in \overline{\mathit{Can}}^{R_1}_e(a, \alpha)$, $\theta \vdash \mathit{Enables}(b, a, \alpha)$, and
- $\forall b \in \overline{\mathit{Can}}^{R_1}_w(a, \alpha)$, $\exists c \in \mathcal{A}$ s.t. $\theta \vdash \mathit{Notifies}(c, b, a, \alpha)$

$R_1 \succeq R_2$ entails $\overline{\mathit{Can}}^{R_2}_e(a, \alpha) \subseteq \overline{\mathit{Can}}^{R_1}_e(a, \alpha)$, which allows us to derive:

- $\forall b \in \overline{\mathit{Can}}^{R_2}_e(a, \alpha)$, $\theta \vdash \mathit{Enables}(b, a, \alpha)$, and
- $\forall b \in \overline{\mathit{Can}}^{R_2}_w(a, \alpha)$, $\exists c \in \mathcal{A}$ s.t. $\theta \vdash \mathit{Notifies}(c, b, a, \alpha)$

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 $R_1 \succeq_\oplus R_2$. As explained above, more privacy for agents means more possibilities to act on their own data and less possibilities for others. □



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399