Computers & Security 144 (2024) 103975

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

A decentralized model for usage and information flow control in distributed systems

Nathanael Denis *, Maryline Laurent, Sophie Chabridon

Télécom SudParis, Institut Polytechnique de Paris, SAMOVAR, Evry, 91120, France

ARTICLE INFO

Dataset link: https://doi.org/10.5281/zenodo.10721222

CCS Concepts:
Computer systems organization: Embedded systems
Redundancy
Robotics
Networks: Network reliability

Keywords:
Modeling
Usage control
Internet of Things
Information flow control
Distributed systems

ABSTRACT

Data usage control enables data owners to enforce policies for their data, by defining authorizations, but also obligations, which are actions to be performed before, during or after being granted access such as accepting web cookies, and conditions bearing on the system and environment attributes, e.g., the time. Usage control is often coupled with information flow control to monitor how data are propagated. While usage control is well established and modeled in centralized systems, the literature has only partially addressed usage control for distributed systems, for instance by distributing the usage control system components. However, when it comes to assigning policy to certain data, it is always enforced by a central authority. This paper proposes an extended usage control model to integrate decentralized information flow control (DIFC), which enables users to decide collectively which policy to apply to their common data. Functions to handle connection status aspects are also considered, for dynamic Internet of Things (IoT) or peer-to-peer networks where parts of the distributed network can be disconnected. Architectural aspects and formal definitions to enable decentralized policies for shared data are proposed as a novelty, resulting from the integration of DIFC. We used the TLA+ formal specification language on the proposed model and its attached model checker TLC to detect potential issues. We detected potential deadlocks due to the new connection functions as well as temporal ordering issues then suggested mitigations accordingly. A privacy analysis is provided using a car-sharing scenario to highlight the benefits of usage control.

1. Introduction

Data usage control enables data owners to enforce policies for their data, by defining authorizations, but also obligations, which are actions to be performed before, during or after being granted access, and conditions bearing on the system and environment attributes such as the time or the system load. Information flow control (IFC) is a security mechanism that regulates the movement of data within a system or network, to prevent undue dissemination of the data (Denning, 1976). A decentralized version called Decentralized Information Flow Control (DIFC), was introduced by Myers and Liskov in 1997. Contrary to traditional IFC, it enables users to define policies individually or as a group without relying on a central entity. Data owners add security labels to the data directly, to indicate who is allowed to read the data (Myers and Liskov, 1997). While usage control and information flow control are two different technologies, they are used jointly by modern usage control systems (UCS) (Harvan and Pretschner, 2009). Therefore, recent models (Kelbert and Pretschner, 2018; Fromm, 2020) tend to encompass both aspects as usage control needs to prevent dissemination in areas outside of the monitoring scope. Introducing usage control can therefore lead to unclear statements, as it is hard to distinguish whether usage control refers to classic usage control (UCON), or classic usage control with information flow control. For disambiguation, the following terminology is used in this article:

- data usage control (UCON): refers to the policies which define the users' rights to access data, based on authorizations, obligations, and conditions;
- information flow control (IFC): refers to the policies about the movement of data within the system, to monitor dissemination;
- usage control: refers to both data usage control and information flow control. It is the general term, corresponding to modern usage control;
- a usage control system (UCS): an entity, centralized or distributed, that is responsible for enforcing usage control.

A formal model of data usage control can help ensure that the system's security and privacy goals are achieved by providing a clear

* Corresponding author.
E-mail addresses: nathanael.denis@telecom-sudparis.eu (N. Denis), maryline.laurent@telecom-sudparis.eu (M. Laurent), sophie.chabridon@telecom-sudparis.eu (S. Chabridon).

https://doi.org/10.1016/j.cose.2024.103975
Received 14 March 2024; Received in revised form 4 June 2024; Accepted 27 June 2024
Available online 3 July 2024
0167-4048/© 2024 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

specification of the access control policies, the resources to be protected, and the roles and responsibilities of the participants in the system. While this modeling has been abundantly addressed in the state-of-the-art in centralized settings (Pretschner et al., 2011; Kelbert and Pretschner, 2013, 2014; Fromm, 2020), it is still incomplete in distributed systems, considering only the distribution of the usage control system (UCS) components, but not of the other system users, notably the policy makers and the data readers (Kelbert and Pretschner, 2015, 2018).

Kelbert and Pretschner proposed a formal model for usage control in distributed systems in 2018 (Kelbert and Pretschner, 2018). It focuses on the distribution of the UCS components, to enable local policy evaluation. In this article, we propose to add the DIFC model (Myers and Liskov, 1997) to this state-of-the-art formalism. DIFC can be used in contexts where data are derived from several sources and all sources must agree to grant or deny access to the data (Myers and Liskov, 1997). It further distributes the system by enabling the definition of policies on data owned by several owners.

Additionally, TLA+ (Temporal Logic of Actions) (Lamport, 2002) is a formal specification language aiming at checking real-time properties. It serves for modeling and analyzing concurrent and distributed systems, making it suitable for validating our proposed model. Specifications in TLA+ are articulated in a formal language of logic and mathematics, to reveal design flaws before system implementation.

In this article, we therefore propose the following contributions:

- the definition of data usage control and information flow control policies in a distributed fashion, further enabling a better consideration of the distribution dimension;
- fine-grained information flow tracking with better granularity as DIFC harnesses application-layer semantics (Liu et al., 2022), which is a known limit of the state-of-the-art (Kelbert and Pretschner, 2018);
- the enrichment of the state-of-the-art formalism for distributed systems (Kelbert and Pretschner, 2018), introducing novel functions and dedicated DIFC components to design and manage decentralized policy making;
- additional functions are introduced to handle the connection status of the network, crucial for unreliable and unstable networks;
- a TLA+ specification of the proposed model on an illustrative car-sharing scenario, to detect misbehaviors, loopholes and corner cases before the implementation phase.

The integration of DIFC is made possible by recent advances in coding practices, highlighted by Liu et al. (Liu et al., 2022). Past approaches to DIFC have required dedicated instrumentation efforts or developer support, which caused little adoption. Nowadays, DIFC can be leveraged by relying on application event logging, a best practice in software development used for telemetry[1] and debugging (Liu et al., 2022).

The elements added to the existing modeling are paramount for a diversity of real-world scenarios. First, Internet of Things networks are usually very dynamic and can be unstable and unreliable, impacting the flow of information and the availability of the data (Ergun et al., 2022). The data generated by Internet of Things devices are often privacy-sensitive, e.g., biometric, and geolocation data, requiring fine-grained policies for data flow. Decentralization plays a key role in the Internet of Things, for both security and performance. Another real-world use case that requires considering connection status is satellite networks. Space is a hostile environment and satellites can disconnect from the network due to physical damage (Fang et al., 2022). Data is also sensitive, in particular for satellite Internet and military use cases.

[1] Telemetry is the in situ collection of measurements or other data at remote points and their automatic transmission to receiving monitoring equipment

The article is structured as follows. First, decentralized information flow control elementaries and the state-of-the-art are provided (Section 2), before introducing the current modeling of usage control for distributed systems (Section 3). We then detail the extension of the existing modeling as our contribution (Section 4). Section 5 gives a TLA+ specification of the proposed model for the car-sharing scenario, to demonstrate how integrators can detect model flaws before the implementation phase. We then analyze the privacy threats in the car-sharing scenario and how usage control helps mitigate these threats. Finally, a concluding section is provided (Section 6).

2. Background on decentralized information flow control

In this section, after the introduction of the DIFC model (Section 2.1), we detail the reasons why decentralized information flow control, while being a canonical approach to access control in systems, was not actually implemented and largely adopted (Section 2.2). Recent advances in software development enable the use of DIFC without additional development costs, which is explained in Section 2.3.

2.1. DIFC model

In addition to information flow control, Myers and Liskov (Myers and Liskov, 1997) proposed a decentralized version of information flow control (DIFC). The main motivation behind DIFC is to provide a security mechanism for distributed systems. The DIFC model allows users to declassify information in a decentralized way, by delegating security to users and groups rather than a monolithic organization or entity, such as the usage control system. In DIFC, the users in the system assign labels to data, and then the system enforces access control policies based on these labels. DIFC labels are defined as pairs of owner-readers p: $\{O : R\}$, where owner O allows the set of readers R to read the information on which the label is attached. The labels are attached to the data by users and are propagated through the system as the information flows from one component to another. Myers and Liskov (Myers and Liskov, 1997) also introduce the following objects, independently of the existing information flow control modeling:

- values that are used in computations;
- slots that are storage locations for values that can take different forms like variables in a code as well as a memory space in the hard drive;
- channels which can be either input channels to read data or output channels to write data. Input channels are read-only slots that allow information to enter the system monitored by DIFC mechanisms, while output channels are write-only slots that serve as an information sink to transmit data outside the system;

In DIFC, labels can be attached to values, but also to slots (and channels which are a particular type of slots).

Operations on labels. In DIFC, the label on a value cannot change, but a new copy of the value can be created with a new label. Myers and Liskov (1997) When this happens we say the value is relabeled even though only the copy has a new label. The key to secure flow is to ensure that any relabeling is consistent with the security policies of the original labeling. Only values can be relabeled, not slots and channels. There are two types of relabeling. Restrictions are relabeling allowing fewer accesses than the original label. Declassifications conversely add more readers to the label. If a label attached to a value cannot change, it is possible to have multiple labels corresponding to different owners.

2.2. DIFC implementation challenges

While decentralized flow control enables the design of data policies whose ownership is shared by several owners, it has several short-

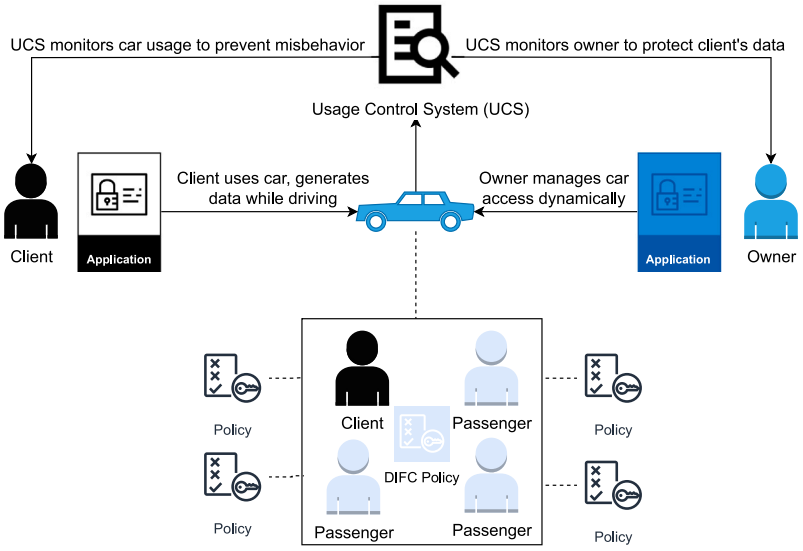


Fig. 1. Car-sharing scenario and policies by car renter and occupants.

comings that severely limit its adoption in real systems (Liu et al., 2022). A major factor limiting the use of DIFC is the tremendous cost in terms of development. DIFC implementations assume that programs are modified to be able to add, remove and handle labels (cf. Section 2.1).

Besides, DIFC implementations can operate at different levels of abstraction like IFC implementations, e.g., operating system level or algorithmic level. DIFC implementations relying on run-time mechanisms are employed at the operating system and programming language levels, which requires additional modifications at this level, making DIFC even more cumbersome (Liu et al., 2022).

2.3. State of the art of DIFC

The above-mentioned limitations to DIFC adoption (cf. Section 2.2) have been addressed indirectly through the development of best coding practices. Indeed, applications now tend to have a security context in the form of application event logs, for debugging, fault detection and telemetry. Events are at the root of information flow control and can be used by DIFC at the application system layer, without requiring additional development costs. Liu et al. (2022) consequently presented their solution T-DIFC. T-DIFC stands for transparent decentralized information flow control and defines DIFC policies based on the inspection of application logging, i.e., using the application layer. Checking policies at the application level saves the administrator from additional development work.

Current best coding practices, leveraged by solutions such as T-DIFC, allow the realistic use of DIFC without the deterrent of additional development work. The current state of the art of DIFC, along with the benefits of DIFC for security, therefore justifies its full integration into the usage control modeling. As of now, DIFC is considered secondary in the usage control modeling, although it is mentioned and considered as a possible addition (cf. Appendix A).

2.4. Car sharing illustrative scenario

As a guided thread for this article, we introduce an illustrative car-sharing scenario where several occupants share the same location data but have different privacy needs. Car sharing (US) or car clubs (GB) is a model of car rental where people rent cars for a short period, often by the hour. They differ from classic rental models in that the owners of the cars are individuals themselves, instead of an agency.

In this scenario, car owners put their cars on a rental market, and car renters pay to rent the cars. The car renter may not be alone in the car. Therefore, car occupants are also considered. The car renter car_{rent}, the car owner car_{own} and the i occupants $\mathcal{O} = \{occ_1, \dots, occ_i\}$ interact with the car-sharing service using a mobile app, considered as trusted. Both the car renter and the car occupants share the same location data, i.e., the location of the car. However, they may have different privacy needs, and the usage control policy should be defined by every one of them, with the most restrictive policy being applied. For instance, an occupant $occ \in$ authorizes the car owner car_{own} to monitor the car, and the car renter car_{rent} authorizes the car owner car_{own} as well as third-party responsible for advertising third-parties adv. If c_{own} logically can access the data, should adv be allowed to access the data as well? This is precisely what DIFC is designed for, as it enables users to define policies as a group by applying labels to data, without requiring a central authority. This scenario is depicted in Fig. 1 and its symbols are summarized in Table 1.

Table 1
Symbols introduced for the car-sharing scenario.

Symbol	Name
car_{own}	Car owner
car_{rent}	Car renter
$\mathcal{O} = \{occ_1, \dots, occ_i\}$	i occupants
c_{car}	A data container in the car
$l_{c_{car}}$	The label associated to c_{car}
d_{GPS}	GPS data associated with a car

3. Existing usage control modeling

The following sections introduce the state of the art on data usage control modeling in distributed systems. First, the data usage control formalism as designed for centralized settings (Section 3.1) is detailed, before the specific aspects of distribution (Section 3.2). We finally detail the rationale behind the need to improve the existing model, to meet IoT requirements regarding the connection status of the network and to integrate DIFC concepts and functions (Section 3.3)

3.1. Information flow control model

An information flow control model is defined by a tuple $(\mathcal{D}, \mathcal{C}, \mathcal{E}, \mathcal{I}, \mathcal{F}, \Sigma)$ (Kelbert and Pretschner, 2018; Fromm, 2020):

- $\mathcal{D}$ are all the data to be protected in the system;

- $\mathcal{C}$ are all representations where the data can be stored in the system and are therefore named containers. For example, containers can represent variables, computer memory cells or a blockchain block;
- $\mathcal{E}$ are events, e.g., method invocations or system calls, that may cause a flow of data and change the system state;
- $\mathcal{I}$ are the principals, all the active entities in a system, i.e., a process or a thread;
- $\mathcal{F}$ is the set of all naming identifiers that are used to identify containers in a system, such as the process-id;
- Σ are all the possible data flow states;

For any set S, $\mathcal{P}(S)$ will denote all the possible subsets of S. For example, if $\mathcal{E} = \{e_1, e_2\}$ is a set of events, $\mathcal{P}(\mathcal{E}) \overset{def}{=} \{\emptyset, e_1, e_2, \{e_1, e_2\}\}$ is composed of all the possible subsets of $\mathcal{E}$. System runs are modeled as a set of timed traces $\mathcal{T} : \mathbb{N} \to \mathcal{P}(\mathcal{E})$ that maps abstract time points t to a set of events $\mathcal{E}$. An event $e \in \mathcal{E}$ itself is composed of a name $e.\text{name} \in \mathcal{N}_\mathcal{E}$ ($\mathcal{N}_\mathcal{E}$ is the set of event names) and a set of parameters $e.\text{p} \in \mathcal{J}_\mathcal{E} \subseteq \mathcal{P}(\mathcal{J})$, $\mathcal{J}$ being the set of all possible parameters. Each parameter $j \in \mathcal{J}$ is in turn defined by a name $j.\text{name} \in \mathcal{N}_\mathcal{P}$ and a value $j.\text{value} \in \mathcal{V}_\mathcal{P}$.

Each event has two special parameters, called $(e.\text{obj}, e.\text{actual}) \in \mathcal{J}_\mathcal{E}^2$. $e.\text{obj}$ denotes the primary object of e, such as a file. The parameter $e.\text{actual}$ is a boolean stating if the event has already happened if true or is only intended if false.[2] Besides, a parameter $e.\text{time} \in \mathcal{J}_\mathcal{E}$ indicates the event's time of observation. Note that $e.\text{time}$ is not a special parameter, as an event may not have been observed yet if it is intended ($e.\text{obj}$ and $e.\text{actual}$ are always defined).

We say that e_1 refines e_2 if their names are the same and if the parameters of e_1 are a superset of e_2. An event refinement is detected using the operation refines $\subseteq \mathcal{E} \times \mathcal{E}$:

$$\forall e_1, e_2 \in \mathcal{E} : e_1 \text{ refines } e_2 \overset{\text{def}}{\iff} e_1.\text{name} = e_2.\text{name} \wedge e_1.\text{p} \supseteq e_2.\text{p} \quad (1)$$

where $e_1.name$, $e_2.names$ are the names of the events e_1 and e_2, and $e_1.\text{p}$, $e_2.\text{p}$ are the parameters of the events e_1 and e_2. Refinement is useful to specify only relevant event parameters and exclude the parameters that are not useful for the evaluation of a given policy, or conversely to gather additional parameters, depending on the use case. As an example, if we consider the following events:

$$e_1 = (send, (obj, mail), (actual, true), (to, \text{“}user@xyz.com\text{”}))$$

and

$$e_2 = (send, (obj, mail), (actual, true), (to, \text{“}user@xyz.com\text{”}),$$

$$(from, \text{“}admin@xyz.com\text{”}))$$

then, we can say that e_2 refines e_1 as the parameters $e_2.\text{p}$ are a superset of $e_1.\text{p}$

The traces $\mathcal{T}$ model system runs by mapping abstract points in time $i \in \mathbb{N}$ to all system events that happened since $i-1$. Traces are formally defined as:

$$\mathcal{T} : \mathbb{N} \to \mathcal{P}(\mathcal{E}), \text{ such that } \forall t \in \mathcal{T}, \forall i \in \mathbb{N}, i > 0,$$
$$\forall e \in t(i) : i - 1 < e.\text{time} \leq i$$

The set of all possible data flow states Σ is defined as $\Sigma \overset{def}{=} s \times a \times n$ where s, a and n are three mappings. Each state $\sigma \in \Sigma$ is defined by

- s is a storage function, $s : \mathcal{C} \to \mathcal{P}(\mathcal{D})$ returns the data stored in a container, capturing which containers potentially store which data;
- a is an alias function, $a : \mathcal{C} \to \mathcal{P}(\mathcal{C})$ which returns the list of containers that may be updated when a given container is updated. It captures the fact that some containers are automatically updated when others do, which is the case when containers store a copy of the same data;
- n is a naming function, $n : \mathcal{F} \to \mathcal{C}$ mapping identifiers to containers.

For each state $\sigma \in \Sigma$, $\sigma.\text{s}$, $\sigma.\text{a}$ and $\sigma.n$ refer to respectively the storage function, the alias function and the naming function defining σ.

With the introduction of data flow states, it is now possible to extend the refines operator to $refines_\Sigma$ which describes the refinement of an event in the presence of a given state. The need for the introduction of $refines_\Sigma$ is that system events always operate on containers $\mathcal{C}$, while policies may be specified in terms of data $\mathcal{D}$. We say that (e_1, σ) refines e_2 if either:

(1) both e_1 and e_2 operate on the same container and if e_1 refines e_2;
(2) if e_1 operates on some container $c \in \mathcal{C}$ and e_2 on some data $d \in \mathcal{D}$ within c and e_1 refines e_2 when ignoring the parameter $e.\text{obj}$.

The formal definition of (e_1, σ) refines e_2 is provided in the appendix for clarity purposes (cf. Appendix B). The different elements, summarized in Table 2 and introduced in this section, enable us to model the information flow. However, it is also necessary to model usage control policies, i.e., how data can be used. Besides, it is necessary to model the reaction of the system when information flow policy violations are detected. We discuss the policy modeling in a dedicated appendix (Appendix A), as it is not compulsory to understand the contributions presented in this article.

Table 2
Symbols related to a distributed system (Kelbert and Pretschner, 2018).

Symbol	Name
$d \in \mathcal{D}$	**Data** (Set of data)
$c \in \mathcal{C}$	**Container** (Set of containers)
$e \in \mathcal{E}$	**Event** (Set of events)
$i \in \mathcal{I}$	**Principals** (Set of principals)
$f \in \mathcal{F}$	**Identifiers** (Set of identifiers)
$\sigma \in \Sigma$	**System state** (Set of system states)
$t \in \mathbb{N}$	**Timestep**
$\mathcal{T}$	**Traces**
$\mathcal{P}(E)$	**Subsets of** E
$e.\text{name} \in \mathcal{N}_\mathcal{E}$	**Event name** (Set of event names)
$e.\text{p} \in \mathcal{J}_\mathcal{E}$	**Event parameter** (Set of parameters)
$j.\text{name} \in \mathcal{N}_\mathcal{P}$	**Parameter name** (Set of parameter names)
$j.\text{value} \in \mathcal{V}_\mathcal{P}$	**Parameter value** (Set of parameter values)

3.2. Existing usage control model for distributed systems

After introducing the general usage control system architecture, we detail the distribution aspects of the state-of-the-art formalism of data usage control. These aspects are threefold. First, the selective distribution of the UCS components. Second, the separation of the whole distributed system into individual systems, that can be merged into sets of individual systems. Finally, the introduction of functions to identify the relevant system sets to evaluate usage control policies.

3.2.1. Usage control system architecture

The Usage Control System (UCS) interacts with Environmental Attributes through Attribute Managers (AM) to recover the values used for data usage control, and with the Controlled Systems themselves. It is composed of the following components, as shown in Fig. 2:

[2] This distinction is important for information flow control, as an intended event can be blocked before the data is disseminated outside the system.

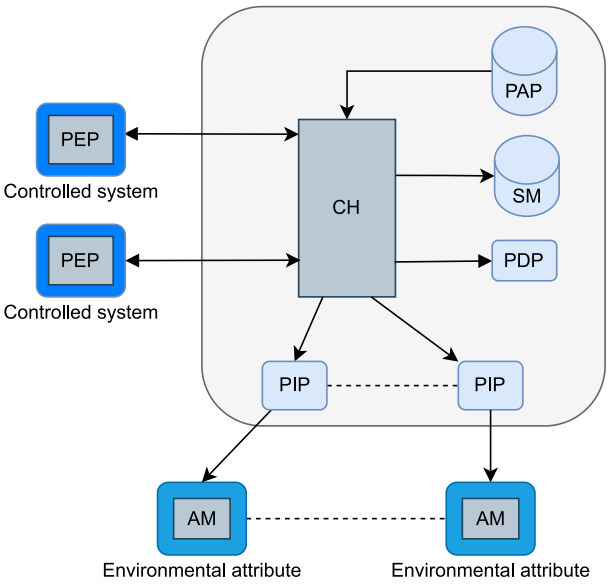


Fig. 2. Usage control framework representation (Rizos et al., 2019).

- PDP: the Policy Decision Point in charge of the policy evaluation. It takes as input an access request, the corresponding policy and the attributes of both users and context. Then it returns the result of the evaluation: Permit, Deny, Undetermined;
- PAP: the Policy Administration Point, which stores the policies;
- PEP: the Policy Enforcement Point enforces the policy evaluation result on a Controlled System;
- PIP: the Policy Information Point, an interface so that the UCS can retrieve the values of the attributes from the system environment;
- CH: the Context Handler, in charge of routing the different processes;
- SM: the Session Manager stores all active sessions and the information needed for monitoring their status.

3.2.2. Splitting the system into individual systems to optimize the policy enforcement

To optimize policy enforcement in distributed systems, it is necessary to identify the relevant set of systems where a given policy must be enforced. This is useful as (1) it avoids monitoring the whole network for every policy; (2) it reduces the network exchanges with the different PDP instances. Individual systems $\mathcal{Y}$ constitute the autonomous parts of the distributed system (Kelbert and Pretschner, 2018). A system is a non-empty set of individual systems whose PEPs share the same PDP/PIP couple. Each individual system $y \in \mathcal{Y}$ has its own containers $\mathcal{C}_y$, its own events $\mathcal{E}_y$, its own set of identifiers $\mathcal{F}_y$, its own possible data flow states Σ_y, and its own system runs $\mathcal{T}_y$. Each event e is required to carry a parameter $e.\text{sys}$ with $\text{sys} \in \mathcal{N}$ ($\mathcal{N}$ is a set of names) to identify the system in which it happens. In practice, individual systems run in parallel and have independent traces and flow states. For policy enforcement, the reasoning must be done on sets of individual systems, called sets of systems $Y \subseteq \mathcal{Y}$ or on the whole distributed system. Sets of systems also have :

(1) a set of events defined as the union of individual system's events: $\mathcal{E}_Y \overset{def}{=} \cup_{y \in Y} \mathcal{E}_y$;
(2) a set of containers $\mathcal{C}_Y \overset{def}{=} \cup_{y \in Y} \mathcal{C}_y$;
(3) a set of identifiers $\mathcal{I}_Y \overset{def}{=} \cup_{y \in Y} \mathcal{I}_y$;
(4) a set of possible system runs (modeled by traces) $\mathcal{T}_Y : \mathbb{N} \to \mathcal{P}(\mathcal{E}_Y)$.

Table 3
Symbols for sets of systems (Kelbert and Pretschner, 2018).

Symbol	Name
y	An individual system
Y	A set of systems
$\mathcal{Y}$	The union of all individual systems
τ	The set of traces of all individual systems
$t_y^\tau \in \mathcal{T}_y$	The trace of the system $y \in \mathcal{Y}$ (all traces of each individual systems)
σ_{t_y}	The data flow state associated to the trace t_y^τ
$t_Y^\tau \in \mathcal{T}_Y$	The combined traces of systems $Y \subseteq \mathcal{Y}$

3.2.3. Relating individual systems with system sets

As part of Kelbert and Prestchner modeling (Kelbert and Pretschner, 2018), it is possible to define the combined traces of systems from individual system traces. Let $\prod$ be the Cartesian product of sets. We can define $\tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y$ as a set of traces of all individual systems. $t_y^\tau \in \mathcal{T}_y$ refers to the trace of the system $y \in \mathcal{Y}$ while σ_{t_y} refers to the corresponding data flow state. $t_Y^\tau \in \mathcal{T}_Y$ is defined as the combined traces of systems $Y \subseteq \mathcal{Y}$. For each timestep $i \in \mathbb{N}$, the set of events observed in Y is the union of the events in all individual systems $y \in Y$. This corresponds to what a centralized PDP would observe.

Similarly, we combine the data flow states of individual systems to form a global data flow state $\sigma_{t_Y^\tau}$, representing what a central PIP would observe. For convenience, $\forall i \in \mathbb{N}$ a given timestep, $\sigma_{t_y^\tau}(i)$ will be noted $\sigma^i_{t_y^\tau}$ in the following. $\sigma_{t_y^\tau}$ is constructed by unifying the mappings of the individual systems' storage, alias, and naming functions.

To recapitulate, t_Y^τ and $\sigma^i_{t_y^\tau}$ of set of systems Y are the reflection of the trace (t_Y^τ) and the state ($\sigma^i_{t_y^\tau}$) of Y. The symbols used for individual systems and sets of systems are summarized in Table 3 and the formal definitions of $t_Y^\tau(i)$, $\sigma^i_{t_y^\tau}$ and $\sigma^i_{t_Y^\tau}.s(c)$ are provided in Appendix B.

3.2.4. Identification functions

To avoid unnecessary communication that would hinder performance, functions are needed to identify the relevant systems for evaluating the policy p. First, the three following auxiliary functions are defined (Kelbert and Pretschner, 2018):

(1) knowC: $\mathcal{P}(C) \to \mathcal{P}(\mathcal{Y})$. Given a set of containers C, knowC returns the set of systems in which one of the containers $c \in C$ resides;
(2) knowD: $\mathcal{P}(D) \times \mathbb{N} \times \prod \mathcal{T} \to \mathcal{P}(\mathcal{Y})$. Given a set of data items D, a point in time $t \in \mathbb{N}$ and a tuple of concurrently executing traces $\mathcal{T}$, knowD relies on the systems' information flow to return the set of systems where one or more containers are storing at least one of the data items $d \in D$. knowD is parameterized in time and requires traces as input, as data are continuously propagated across systems;
(3) happens: $\mathcal{E} \times \mathbb{N} \times \prod \mathcal{T} \to \mathcal{P}(\mathcal{Y})$. Given an event $e \in \mathcal{E}$, a point in time, and a tuple of concurrently executing traces, happens returns the set of systems where any event refining e happens.

Using these three auxiliary functions (cf. Appendix B for the full formal definitions), the relevant function is composed so as to detect individual systems potentially interesting for the evaluation of policies. For a policy p, relevant: $\Phi \times \mathbb{N} \times \prod \mathcal{T}$ identifies the set of systems that may contribute to the evaluation of conditions $\varphi_p \in \Phi$ at a given timestep $i \in \mathbb{N}$ and given a set of concurrently executing traces $\tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y$. relevant is defined using the operators Ψ, Ω, Φ *and* Γ (cf. Fig. 7, Appendices A and B).

3.3. Rationale for extending the existing model

Distributed systems are computer systems composed of multiple independent components, such as processors, storage devices, and communication channels, that cooperate to perform a unified task. Their purpose is to provide high performance, scalability, fault tolerance,

and transparency to the users. They are consequently used in several fields including the Internet of Things (IoT), Cloud computing and peer-to-peer networks.

The current state-of-the-art formalism for usage control in distributed systems of Kelbert and Pretschner (Kelbert and Pretschner, 2018) (cf. Section 3.2) indeed considers the distribution of the UCS components, notably the PDP/PIP couple. However, the dimension of user-side distribution is important, where groups of data owners define possibly conflicting policies for access. We propose to use the DIFC to define these group policies. In general, data usage control in distributed systems is a recent field of research, and actual challenges that must be addressed are not clearly identified nor formalized (Gil et al., 2022).

The possibility of integrating DIFC in the usage control model is mentioned in the state of the art, claiming that (1) data usage control policies can express similar constraints to DIFC; (2) DIFC mechanisms can be integrated via the evalExt operator. While the former affirmation is partly true regarding the constraints in the policies, data usage control policies do not consider the owner-side distribution of the data, and ignore the architectural consequences for the UCS. For the second affirmation, the evaluation of external specification is indeed considered, but the evalExt operator requires additional analyses when a concrete framework realizing evalExt is implemented. These limitations, the need to consider distribution as a whole and the recent advancements in DIFC enabling its practical use justify the integration of DIFC in distributed usage control modeling.

Besides, individual systems as defined in Section 3.2 may be disconnected from the network at a given timestep $i \in \mathbb{N}$. This is particularly true in the Internet of Things, as devices can be intermittent, on battery, or unreliable. The formalism must be able to take into account the possibility that a data d is present in a container, but that the system where the container is located cannot be reached at a given timestep.

To sum up, the extension of the existing model is justified by:

(1) the need for users to define policies jointly in a decentralized manner. This requires the introduction of DIFC in the existing formalism, as the $evalExt$ operator is too generic to ensure the correctness of the solution;
(2) the need to consider the connection status of individual systems.

4. Proposed extension

In this Section 4, we introduce novel auxiliary functions and dedicated components to evaluate DIFC policies in addition to data usage control policies. This results in different functions, where the focus is put on detecting conflicts between labels. Additionally, we define functions to monitor the status of a system and determine if data and containers are currently accessible.

We present an illustrative car-sharing scenario where several occupants share the same location data but have different privacy needs. Car sharing (US) or car clubs (GB) is a model of car rental where people rent cars for a short period, often by the hour. They differ from classic rental models in that the owners of the cars are individuals themselves, instead of an agency.

In this scenario, car owners put their cars on a rental market, and car renters pay to rent the cars. The car renter may not be alone in the car. Therefore, car occupants are also considered. The car renter car_{rent}, the car owner car_{own} and the i occupants $\mathcal{O} = \{occ_1, \dots, occ_i\}$ interact with the car-sharing service using a mobile app, considered as trusted. Both the car renter and the car occupants share the same location data, i.e., the location of the car. However, they may have different privacy needs, and the usage control policy should be defined by every one of them, with the most restrictive policy being applied. For instance, an occupant $occ \in$ authorizes the car owner car_{own} to monitor the car, and the car renter car_{rent} authorizes the car owner car_{own} as well as third-party responsible for advertising third-parties adv. If c_{own} logically can access the data, should adv be allowed to access the data as well? This is precisely what DIFC is designed for, as it enables users to define policies as a group by applying labels to data, without requiring a central authority. This scenario is depicted in Fig. 1 and its symbols are summarized in Table 1.

4.1. Additional functions for DIFC and network status aspects

In this section, new functions are introduced to the existing modeling. First, we adapt the DIFC labels to the usage control model and define new functions accordingly to handle labels in the distributed system. Second, we define functions to determine the connection status of individual systems in the network, for real-life use cases, e.g., IoT networks. The introduced functions are summarized in Table 4

4.1.1. Functions for data and container labeling

As DIFC is only mentioned as an external specification in the state-of-the-art, DIFC objects, namely values, slots, channels and labels are not formalized in usage control models per se. However, it is possible to align DIFC objects with usage control concepts, except for labels. Data $d \in \mathcal{D}$ can indeed be regarded as DIFC values. Similarly, as containers hold the data, they can be considered as slots[3] as defined in the DIFC specification and have labels attached to them. A set of labels is noted $\mathcal{L}$ and an individual label is noted l. Equivalently and for simplification, we associate labels to usage control concepts rather than DIFC concepts, namely containers C and data D. For convenience, we keep track of the objects associated with a label, such that $l.containers$ and $l.data$ return the list of, respectively, containers and data on which the label is applied. Therefore, the formal definition of labels is the following:

$$l = \{p : \{O : R\}, C, D\}$$

where $p : \{O : R\}$ is a pair of owner-readers where owner O allows the set of readers R to access the container.

Having different labels, data and containers may have conflicting policies. Detecting conflicts is crucial to enforce information flow policies, as a decision must be taken to prioritize either the container labeling or the data labeling. Detecting the conflicts requires additional functions, to fetch the labels associated with containers and data, which are defined here as part of the contribution (the formal definitions are provided in Appendix C):

(1) labelC: $\mathcal{P}(C) \rightarrow \mathcal{P}(\mathcal{L})$. Given a set of containers, returns the labels associated with the containers: $\forall c \in C$, $\text{labelC}(c) \stackrel{def}{=} \{l \in \mathcal{L} \mid c \in l.containers\}$;
(2) labelD: $\mathcal{P}(D) \rightarrow \mathcal{P}(\mathcal{L})$. Given a set of data, returns the labels associated with the data items: $\forall d \in \mathcal{D}$, $\text{labelD}(d) \stackrel{def}{=} \{l \in \mathcal{L} \mid d \in l.data\}$
(3) conflict: $\mathcal{P}(C) \times \mathcal{P}(D) \rightarrow \mathcal{P}(\mathcal{Y})$. Given a set of containers C and data $\mathcal{D}$, returns the set of systems where there is a conflict between the containers and the data that a user is allowed to access. conflicts detects if there is at least one data item in d such that at least one reader r ($p \neq \emptyset$) is authorized to access d, but there is a container $c \in C_y$ knowing d which is forbidden to r. This function relies on knowD (cf. Appendix B) to identify the containers C knowing d in the first place: $\forall c \in C, \forall d \in \mathcal{D}$, $\text{conflict}(c, d) \stackrel{def}{=} \{y \in knowD(d) \mid \exists l \in \mathcal{L}, l \in labelD(d) \wedge l \notin labelC(c)\}$;

The conflict function, based on the labelC and labelD auxiliary functions, is dedicated to detecting individual systems $\mathcal{Y}$ where there is a conflict between the data label and the label of the container holding the data. These systems are of interest because this situation requires

[3] Containers can therefore be considered as DIFC channels, channels being read-only or write-only DIFC slots

Table 4
Functions of the formal model.

Function	Parameters	Purpose
connected	A set of systems Y, a point in time t and a set of traces $\mathcal{T}$	Connection status
availableC	A set of containers C, a point in time t and a set of traces $\mathcal{T}$	Container availability
availableD	A set of data items D, a point in time t and a set of traces $\mathcal{T}$	Data availability
knowC	A set of containers C	Find containers' location
knowD	A set of data items D, a point in time t and a set of traces $\mathcal{T}$	Find data location
happens	An event $e \in \mathcal{E}$, a point in time t , and a set of traces $\mathcal{T}$	Event occurrence
labelC	A set of containers C	Return container label
labelD	A set of data D	Return data labels
conflict	A set of containers C and data D	Discrepancy between labels

deciding which label should be prioritized, e.g., by considering the label representing the most restrictive policy.

In the car-sharing scenario, a car owner car_{own} is allowed to access the data container in its car c_{car}, but may not have access to the location of the car d_{GPS} stored in c_{car}. The car renter car_{rent} or one of the car occupants $\mathcal{O}$ may apply labels to d_{GPS} that may prevent the car_{rent} from reading d_{GPS}. Detecting a conflict between the label $l_{c_{car}}$ applied to the container c_{car} and the label $l_{d_{GPS}}$ applied to the data d_{GPS} enables to block the dissemination to car_{own}. In that case, the security policy is to consider that the data label has priority over the container label. This policy is defined by the system administrator and could be different, e.g., deny access in case of conflict is detected which would have the same result in this example.

Practical implications. The introduction of a decentralized information flow control mechanism, instead of its by-default centralized counterpart, removes a single point of failure from the system. Indeed, a centralized IFC can be malicious, corrupted, unreliable, and grant itself or any agent access to virtually all data in the network. The centralized agent responsible for policymaking can also be the target of denial of service attacks, which would render the whole network inoperant. The use of DIFC for data confidentiality and integrity already has applications in cloud environments (Lu et al., 2022) and is relevant for the Internet of Things where decentralization is a crucial aspect of performance, security and privacy (Skarmeta et al., 2014).

4.1.2. Functions for network status

Distributed systems are used in a diversity of networks, including the Internet of Things and peer-to-peer networks. These networks are inherently dynamic, with devices and peers connecting and disconnecting very frequently. Besides, the network may be intermittent or unreliable. For proper policy evaluation, it is consequently required to determine whether an individual system or a set of systems $\mathcal{Y}$ are currently connected to a system. More accurately, an individual (or a set of) system is considered disconnected if it cannot be reached by any individual system in the usage control system. The main concern in our case is the possibility of evaluating policies by accessing the data containers. By extension, it makes it possible to identify the subset of systems $\mathcal{Y}$ where a container C resides and that is connected to the distributed network. This leads to the definition of the following functions (formally defined in Appendix C):

(1) connected : $\mathcal{Y}\times\mathbb{N}\times\prod\mathcal{T} \rightarrow \mathcal{P}(\mathcal{Y})$. Given a set of individual systems $\mathcal{Y}$, a point in time $i \in \mathbb{N}$ and a tuple of traces, return the subset of individual systems that are reachable in the network at this point in time. Traces are needed because the connection status depends on the timesteps considered. An individual system y is considered connected if there is no disconnection $e_{disconnect}$ in the traces, i.e., an event whose name e.name = disconnect.
(2) availableC : $\mathcal{P}(C) \times \mathbb{N} \times \prod\mathcal{T} \rightarrow \mathcal{P}(\mathcal{Y})$. Given a set of containers, a point in time and a tuple of concurrently executing traces $\mathcal{T}$, availableC returns the set of currently reachable systems in which one of the containers $c \in C$ resides.
(3) availableD : $\mathcal{P}(D)\times\mathbb{N}\times\prod\mathcal{T} \rightarrow \mathcal{P}(\mathcal{Y})$. Given a set of data items $\mathcal{D}$, a point in time $t \in \mathbb{N}$ and a tuple of concurrently executing traces $\mathcal{T}$, availableD returns the set of currently reachable systems where one or more containers are storing at least one of the data items $d \in \mathcal{D}$.

As the cars are moving objects, the need to manage connection status is paramount in the car-sharing scenario. First, a car may be disconnected temporarily from the network, e.g., driving through a tunnel, which is detected by the connected function. As a consequence, if the location of the car d_{GPS} is stored only via a data container c_{car} in the car, it becomes unavailable to the car owner car_{own}, who should be notified. This is reflected by both availableC and availableD, for respectively the container c_{car} and the data d_{GPS}.

Practical implications. Adding functions to handle the connection status of individual systems of the network serves a diversity of use cases, where connection can be unreliable. Internet of Things applications are characterized by their dynamicity (Li et al., 2014), as devices can connect and disconnect from the network and their data containers become unreachable. It is also beneficial when the integrity of the data containers can be threatened, in which case the disconnection is permanent. This is the case for hostile environments including military applications, satellite communications and a variety of crises including cyber attacks (Alvarez et al., 2016).

In this section, we introduced several formal functions to handle labels (labelC and labelD) and detect conflicts between the data and their containers (conflict). Additionally, we introduced functions to handle the status of the network, detecting if a set of individual systems is connected (or not connected), and consequently if a container can be accessed and data fetched depending on the system's state. In the next section, we introduce the new components needed in the usage control system architecture to handle DIFC aspects, as part of our contribution. Table 4 provides all the functions introduced in this article, as well as their parameters and their purpose.

4.2. Integrating DIFC components into the usage control system

Distributing the system requires the introduction of several components for the UCS to handle cross-system communication, information flow tracking and policy deployment. In addition to the existing UCS components in centralized settings (cf. Section 3.2), the usage control system has the following components:

- the Distribution Management Point (DMP) organizes the information flow between systems and the policy propagation while relying on a distributed database to coordinate the policy decisions;
- the Policy Management Point (PMP) deploys the ECA rules to be enforced at the Policy Decision Point (PDP);
- the PDP server provides the interface so that PEPs can request data usage control decisions.
- the PIP Server provides the interface to fetch environment attributes from the PIPs.
- the DMP Server allows communication with remote DMPs.

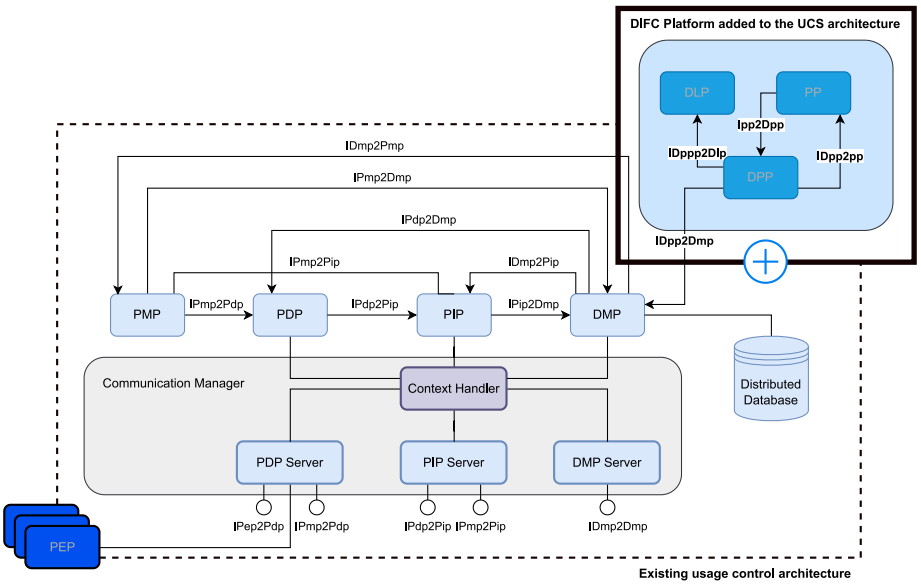


Fig. 3. Proposed augmented architecture (Kelbert and Pretschner, 2018) with the DIFC feature for policy labeling. The Data Labeling Point (DLP), the Privilege Point (PP) and the Data Processing Point (DPP) constitute the added DIFC platform instance.

- the Context Handler orders and processes requests sent to the servers, notably to avoid event reordering and race conditions. The architecture is depicted in Fig. 3;
- the communication manager manages all external communications and runs servers to show the components' functionalities to the outside. It is composed of the Context Handler, PDP server, PIP server and DMP server.

4.2.1. Additional components for DIFC processing

Decentralized information flow control is often processed using a dedicated instance, called the DIFC platform (Schultz and Liskov, 2013) or DIFC center (Lu et al., 2022). There are different ways to process DIFC tasks: either use already existing components, e.g., the PDP, or introduce new dedicated components to manage DIFC aspects. Since DIFC policies differ from usage control policies, we introduce dedicated components to handle DIFC aspects.

For distribution purposes, DIFC platform components are added as a novelty to the existing architecture (Lu et al., 2022), and interact only with the DMP to distribute the policies to other individual systems $\mathcal{Y}$ or system set Y. The DIFC platform is responsible for data pre-processing (DPP), privilege extraction (PP) and data labeling (DLP). Each operation has a dedicated component, as depicted in Fig. 3:

- the Data Pre-processing Point (DPP) routes the data to other components of the DIFC platform. If the data are unlabeled, the data are sent to the DLP. Conversely, labeled data are sent to PP for privilege extraction (cf. Section 2.1);
- the Privilege Point (PP) is responsible for privilege extraction, converting labels into actual access rights. Consequently, it is also responsible for detecting conflicts between labels to be able to determine access rights;
- the Data Labeling Point (DLP) is responsible for labeling data and containers.

4.2.2. Existing interfaces

Interfaces are associated with the components to provide several methods for usage control. The PDP server provides two interfaces `IPmp2Pdp` and `IPep2Pdp`, providing methods to deploy or retrieve policies, to signal system events and to await policy decisions. The PDP uses interface `IPdp2Pip` to evaluate state-based operators, retrieve all data stored in a container C, and signal system events associated with information flow. The PMP uses interface `IPmp2Pip` to inform the PIP about initial data representations. Interface `IDmp2Pip` is used by the DMP if there are cross-system data flows. The DMP provides interface `IDmp2Pmp`, which is used by the local DMP to deploy policies and to retrieve currently deployed policies. The DMP additionally provides four interfaces:

(1) `IPip2Dmp` allows the PIP to inform the DMP about data flows to the remote socket containers;
(2) `IPdp2Dmp` enables the coordination of policy decisions across PDPs. This includes methods to notify that an operator's state has changed, to query whether the state of some operator has changed at remote PDPs, and to synchronize the points in time in which policies are evaluated;
(3) `IPmp2Dmp` allows the PMP to register policies at the DMP;
(4) `IDmp2Dmp` provides functionalities for cross-system information flow tracking and policy propagation between remote DMPs.

4.2.3. DIFC interfaces

As part of the contribution, the introduction of DIFC components requires four additional interfaces (shown in Fig. 3):

(1) `IDPP2DLP` is an interface implemented so that the data can be labeled by the data labeling point (DLP) if the component DPP detects that it has not been labeled yet;
(2) `IPP2DPP` is needed to answer a request from the data processing point (DPP) and to notify it with updates of privileges;
(3) `IDPP2PP` is an interface that enables the data pre-processing point (DPP) to fetch the required privileges from the dedicated privilege point;
(4) `IDPP2DMP` provide methods so that the DIFC policies are propagated to all individual systems.

4.2.4. Distribution of the DIFC platform

To enable further consideration of the distribution dimension, and to avoid introducing a single point of failure, we also consider the distribution of the DIFC platform itself. In Section 3.2, we detailed how several PDP/PIP couples are deployed locally, to reduce the time needed to make a policy decision. The same principle can be applied to the DIFC platform, which is composed of different DIFC instances. These instances communicate the changes in DIFC policies, i.e. labels, using the Data Management Point (DMP). The DIFC policies are then

deployed at the local PDP using the interface `IDmp2Pmp` like usage control policies.

5. Evaluation of the proposed model

To ensure the correctness of the proposed modeling, we rely on TLA+ specification and model checking using its associated model checker TLC, introduced in Section 5.1. Model checking is used in formal verification for early detection of errors or potential issues in a system design or specification, for an exhaustive exploration of all system states and behaviors. Additionally, we provide an analysis of the privacy threats in the car-sharing scenario and describe the mitigations provided by the usage control system. This privacy analysis highlights the benefits of usage control for privacy preservation and the risks in the car-sharing scenario, that may serve as a basis for other similar scenarios.

5.1. Background on TLA+ and TLC

TLA+ is a general-purpose formal specification language based on the Temporal Logic of Actions for specifying real-time properties (Lamport, 1992). It is designed for modeling and reasoning on concurrent and distributed systems, which makes it appropriate to check our proposed model. TLA+ specifications are written in a formal language based on temporal logic and mathematics, and the precision of specifications written in this language is intended to uncover design flaws before system implementation.

TLA+ model checker (TLC). Alongside TLA+, TLC (Temporal Logic of Actions Model Checker) is a model checker for verifying the properties of systems specified using TLA+ (Temporal Logic of Actions). TLC allows one to check whether a TLA+ specification satisfies various properties, such as safety and liveness, by exhaustively exploring all possible states and behaviors of the specified system. Liveness properties state "that the program eventually enters a desirable state" (Owicki and Lamport, 1982). Examples of liveness properties in the car-sharing scenario are: (1) each access request eventually leads to a usage control decision, either accepted or denied; (2) a user who has been granted access will eventually access the corresponding data. Conversely, safety properties state "that the program never enters an unacceptable state" (Owicki and Lamport, 1982). Examples of safety properties in the car-sharing illustrative scenario are: (1) if a car renter paid to access the car, he or she never ends up in a state where the car remains locked and there is no refund; (2) if a car renter or one occupant does not allow the tracking of the car's location, the system should not allow the car to start and share the location.

TLC error traces. When TLC finds a counterexample that violates a specified property, it provides an error trace, which is a sequence of states and actions that lead to the violation. This trace helps in diagnosing and understanding the issue in the system. In particular, TLC checks for:

- silliness errors: mathematical and logical errors, e.g., division by zero, dividing an integer by a string...
- deadlocks: when the system cannot enter the following state, referred to as Next in TLA+;
- user-specified properties: validation of safety and liveness properties.

TLC has two different modes to detect errors. First, the model check mode explores all possible states. This mode can be used when the number of states is finite, which is not true for all specifications. Second, the simulate mode explores randomly generated behaviors. Fig. 4 summarizes the TLC workflow.

5.2. Rationale for specification

Writing a specification is a complex, time-consuming process and must be justified accordingly. According to Leslie Lamport, "the purpose of writing a specification is to help avoid errors" (Lamport, 2002) in the following ways:

(R1) help the design process, notably by detecting problems and subtle interactions in the early design phase, rather than the implementation phase;
(R2) provide a clear way to communicate a design for the system;
(R3) find errors in the design and test the system, in particular with the TLC model checker.

The main contribution of this article is the introduction of new formal functions to handle DIFC aspects and connection status (cf. Section 4.1). These new functions may introduce corner cases or have unexpected behaviors, e.g., what happens if a conflict is detected in a disconnected individual system? Detecting these issues (R1) is paramount to the proper functioning of the distributed systems and the functions introduced. Testing the implementation using TLC (R3) is also a motivation to help find errors in the design, but more interestingly in the model itself.

5.3. Car-sharing interactions

To model the system's interactions with better accuracy, we have written a TLA+ specification on the car-sharing illustrative scenario (cf. Section 2.4). We detail here the interactions between the system's agents, i.e., the car, the car renter, the occupants, the car owner and the components of the UCS (cf. Fig. 3). The whole renting process is considered, in particular the car renter's access to the car, the joint policy definition, and the distributed usage control management. The system and network calls corresponding to these interactions are depicted in Fig. 5.

Access sequence. First, the car renters require access to the car by sending an `accessRequest` call to the usage control system sent by the policy enforcement point (PEP) to the PDP server. The car renters and the occupants define their own policies regarding location, sending successive `sendDIFCPolicy` calls to the PDP server as well. The PDP server forwards the policies to the context handler (CH) (`forwardDIFCPolicies`), then the CH routes the policy to the DIFC platform using the distribution management point (DMP) with a `requestLabeling` call. The DMP transfers the `requestLabeling` call to the data pre-processing point (DPP), which in turn forwards it to the data labeling point (DLP) as the location data are not labeled yet. The DLP labels the location data with the given DIFC policies (`label`), then returns the labeled data to the DPP (`returnLabeledData`). The DPP asks the privilege point (PP) to compute the compound DIFC policy (`computeDIFCpolicy`) resulting from individual labels, i.e., the labels of the car renter and its occupants. Using the new label, the privilege point extracts the car owner's access rights, which are returned to the DPP (`returnAccessRights`). The decision is forwarded to the DMP for propagation across the individual systems, then to the context handler, and finally to the PDP server. The PDP servers send the access decision (`sendDecision`) computed from the access rights to the PEP on the car for enforcement. If the car renter and its occupants agree to share their location, the car is unlocked or remains locked otherwise (both using an `enforceDecision` call). Before the car owner receives the car location from the PIP, the `conflict` call (cf. Section 4.1) is sent from the CH to the DMP to ensure the car owner can access both the data and the container holding them. The DMP returns the set of systems where a conflict occurs; if there is no conflict detected (`conflictResult` sent from the CH to the PEP), then the car owner is granted access to the car location by the PEP using `sendGPSData`.

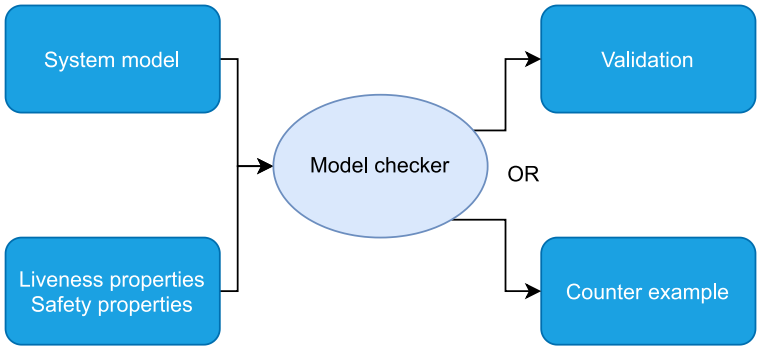


Fig. 4. TLC model checker - Workflow.

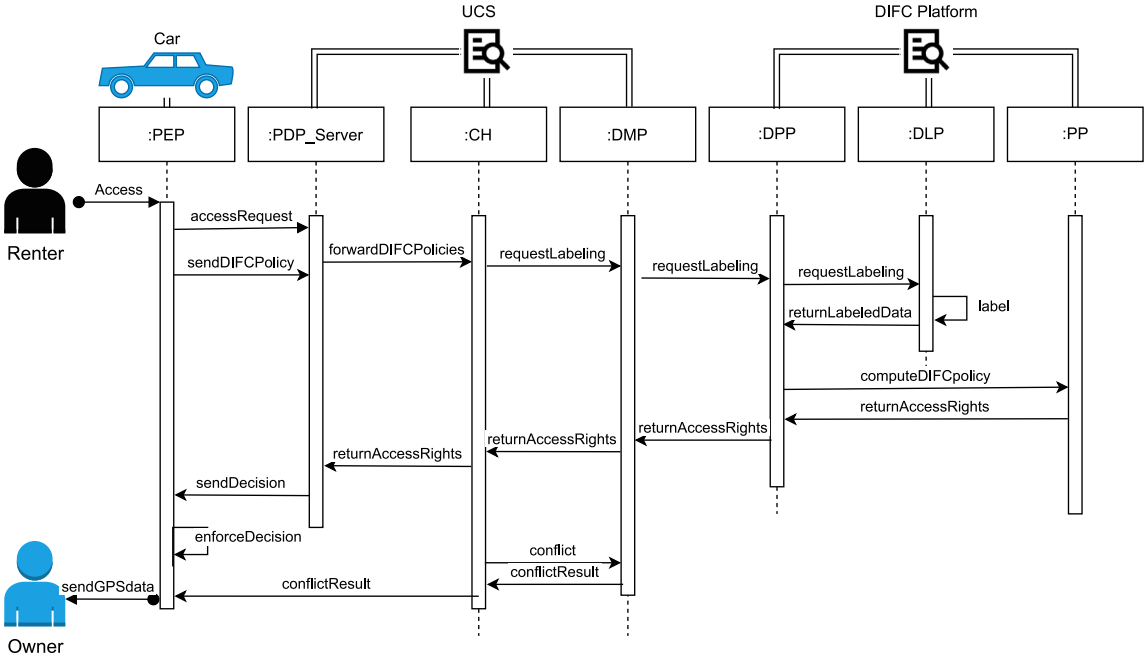


Fig. 5. Sequence diagram leading to an access in the car-sharing illustrative scenario.

Once the car owner has access to its car location, it must process it in compliance with the policies of the car renters and the occupants. The sequence diagram for data policy compliance evaluation is provided in Fig. 6. The usage control system is responsible for the monitoring of the data usage. To this end, the usage control system fetches the car owners' attributes (`fetchAttributes`) using the local policy information points (PIPs). First, the usage control system must identify the most relevant PDP/PIP couple, which first requires to identify the individual system where the car location can be fetched, sending `availableD` to the DMP (cf. Section 4.1). The closest PIP instance to the identified system sends the attributes to the context handler (CH) (`sendAttributes`) which in turn routes the attributes to the coupled policy decision point (PDP). The PDP instance evaluates the usage control policies (`evaluateUsageControl`) and then sends the evaluation results to the context handler (`sendResult`). The context handler forwards the results to the policy enforcement point, which puts an end to the access if the policy is violated.

5.4. Lessons learned

There are potential border effects and potential deadlocks related to the new functions introduced in the formalism, that can be detected using TLA+ and need to be considered when implementing the model. In this section, we identify the potential sources for TLC warnings (cf. Section 5) and how to mitigate them.

5.4.1. Labeling functions

The Context Handler orders and processes requests sent to the servers and is responsible for avoiding event reordering and race conditions. In the newly introduced DIFC Platform, there is no component responsible for event ordering, and the following issue may arise: at a given timestep i, the DIFC platform receives a first event $e_{relabel}$ bearing the label l_{new} to be attached to the data d, and another event e_{eval} requiring to extract the privileges for the data d currently labeled with l_{old}. It is paramount for security so that $e_{relabel}$ is processed by the platform before e_{eval} since the result of e_{eval} will differ after the relabeling. It is possible to detect this kind of behavior in TLA+ using temporal ordering. The temporal property and the associated functions for relabeling and privilege extraction are specified in Listing 1.

5.4.2. Interactions between labeling and connection status functions

Labeling functions can be negatively impacted by the connection status of a system, which creates several troublesome situations.

(1) The conflict function checks that a data d and the container c holding that data do not have conflicting labels, i.e., access policies. However, to fetch the labels of a container and the data within, using respectively labelC and labelD, the container and its corresponding system must be reachable. Trying to detect conflicts while the container is in a disconnected individual

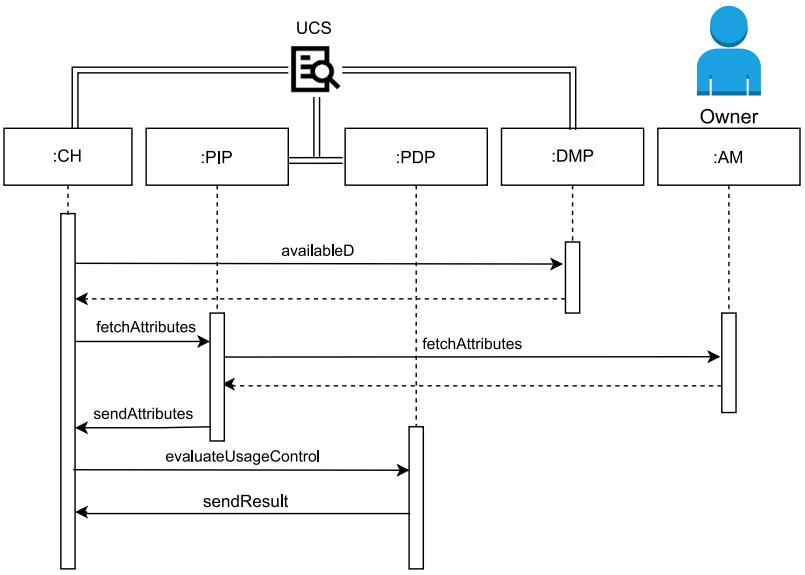


Fig. 6. Sequence diagram for data policy compliance evaluation in the car-sharing illustrative scenario.

```
L = { << "L1", { << "O1", {"R1", "R2"} >> }, {"D1", "D2"}, {"c1"} >>}
\* Example label for clarity
relabeling(data, newlabel) ==
    newlabel[3] = data

extract_privilege(label, user, data) ==
    user \in label[2][2] /\ data \in label[3]

TemporalOrdering ==
    \A data, newlabel, user:
        /\ relabeling(data, newlabel)
        ~> extract_privilege(newlabel, user, data)
```

Listing 1: Event ordering management for relabeling and privilege extraction in TLA+ Code

system triggers a TLA+ error since the system ends up in a loophole without a reconnection. The mitigations to this issue when implementing include (1) consider that there is a conflict by default as a strict policy; (2) define an unknown answer to the conflict function, then call the knowC and connected functions to notify if other containers hold the data d.

(2) The previous statement can be extended to the existing functions, i.e., knowD, knowC and happens. As an example, if the function knowC is called upon a disconnected system $y_{disconnected}$, the system ends up in a loophole as $y_{disconnected}$ does not answer. A possible mitigation is to share $y_{disconnected}$ status with the system via its network peers.

A potential mitigation to the impossibility of detecting conflicts in a disconnected system is proposed in Listing 2.

By specifying the car-sharing scenario using TLA+, we successfully detected general misbehaviors of the system triggered by the introduction of the connection status functions and the labeling functions. In particular, TLC detected silliness errors (1) and deadlocks (2) when trying to access data or conflicts in a disconnected container (cf. Section 5.1). We suggested several mitigations as examples to help with future implementations.

5.5. Privacy analysis

Our primary focus is to enforce compliance with usage control policies and thwart attempts to acquire unprotected data duplicates. We address in this section privacy threats to highlight how usage control can help mitigate them.

5.5.1. Privacy analysis

The privacy threat assessment is conducted on the car-sharing scenario (Section 2.4), to describe the threats more accurately. The analysis can nonetheless be generalized to any use case requiring data usage control and information flow control. The LINDDUN privacy threat modeling framework (Deng et al., 2011) is used to describe extensively the potential threats to privacy and help to frame the analysis.

Items of interest. Items of interest (IOIs) in LINDDUN refer to any data element, including users' personal data or transaction data, that may be privacy-sensitive. IOIs can be subjects, messages, actions or data. In the car-sharing scenario:

- the car location;
- the transaction data, when renting the car;
- users' data, i.e., personal data of car owners, car renters and passengers;
- usage control data, such as the results of access requests;
- metadata, such as the time the location data were created which can be used to infer other data.
- network messages between the participants, and the messages between the UCS components;
- actions carried out by the participants, such as an access request to the car by the client or the beginning of an access request evaluation by the UCS.

```
unknownSet == {-1} \* To represent unknown and keep type coherence

conflict(l1, l2, y, i, t) ==
    IF connected(y, i, t) = {}
        \* Connected is empty if y is not connected
        THEN {-1} \* Return "unknown" if the set is disconnected
        ELSE l1 = l2

findBackup(c1, d1, y, i, t) ==
    IF conflict(c1, d1, y, i, t) = "unknown"
        THEN knowD(d1, i)
        ELSE unknownSet
```

Listing 2: Unknown Set and Conflict Resolution in TLA+

Linkability (L). Linking occurs when two items of interest are associated to learn more about an individual or a group. In the car-sharing scenario, it is possible to link a car renter and a car owner. It simplifies re-identification when one of them is identified (I). Inferring the location of a car renter also increases the probability of locating the associated car owner, e.g., both live in the same city. Linking leads to other identifying (I), detection (D) and non-repudiation (N) threats and is triggered by data disclosure (D), i.e., the likelihood of linking threats as more data is disseminated.

Identifiability (I). Identification occurs when an attacker learns the identity of an individual, breaking its anonymity or pseudonymity. This threat distinguishes identified data where the identity is explicitly maintained, and identifiable data which enables the derivation of identities. Users' attributes such as IP addresses can be used to infer the user's true identity. In the scenario, the risks are to re-identify car owners, car renters and passengers, using for instance car location data, usage control data or metadata.

Non-repudiation (N). Non-repudiation in LINDDUN privacy threat assessment is the ability to attribute claims to an individual. For example, the impossibility for car renters to deny they rented and accessed a car, or for car owners to deny they rented their car to a given car renter, are non-repudiation threats.

Detectability (D). Detecting is the ability to deduce the involvement of an individual through observation. Detecting does not require the actual data value. Knowing the existence of data is enough to infer more sensitive information. Detecting can be done by observing communications directly or through application side-effects. Detecting threats in the car-sharing scenario include:

(1) detecting that a user is monitored by analyzing the communications of the UCS. An attacker learns that the user has likely bought valuable data such as the location of the car;
(2) detecting users' attributes by intercepting the communications between the policy information points (PIP) and the context handler (CH) (cf. Section 3.2.1);

Data disclosure (D). Data disclosure is the immoderate collecting, storing, processing or sharing of personal data. This generic threat focuses on four characteristics:

(1) unnecessary data types, if the data granularity or sensitivity is better than required;
(2) excessive data volume, if the data processing is too frequent;
(3) unnecessary processing, if the data is processed or disseminated out of necessity;
(4) excessive exposure, if the data is widely accessible to irrelevant users.

In the car-sharing scenario, data disclosure can occur if a car owner has access to the location data while not fulfilling the policy's conditions, or can disseminate it outside of the UCS scope. Similarly, data disclosure occurs if the UCS collects too much detailed data about the car owners to monitor them.

Unawareness (U). Unawareness occurs when an individual is insufficiently informed and involved in the processing of personal data. Unawareness threats exist in the scenario if the car renters and their passengers are not informed about the privacy threats they face by renting the car and sharing their location.

Non-compliance (N). Non-compliance is the deviation from security and data management best practices, standards and legislation. This risk occurs if the processing of any item of interest is considered unlawful based on the specified policies and the applicable regulations, e.g., GDPR (The European Parliament and the Council of the European Union, 2018). In the car-sharing scenario, non-compliance threats are that although a car renter and its passengers specify a usage control policy, this policy is not properly enforced by the UCS.

5.5.2. Privacy mitigations

Usage control is designed to address several families of privacy threats. We next detail how these threats are addressed, and which ones cannot be mitigated by the UCS. Table 5 summarizes the threats mitigated by the UCS.

Linkability ∼. The usage control system has only a partial impact in preventing this threat. The UCS reduces the linkability threat probability by limiting the dissemination of the location, which could be used to link the car renter and the car owner. However, the UCS is not meant to address linkability directly.

Identifiability ∼. The usage control system must be able to identify the car owner for monitoring purposes. While it can the car owner from disclosing the car renters' identity – or its own identity unintentionally –, it is much harder for the USC to prevent inference from identifiable data.

Non-repudiation ✗. To fulfill its task, the UCS needs to ensure that a monitored car owner cannot decline having disseminated the data, or having processed it unlawfully. Therefore, the UCS does not guarantee non-repudiation and even makes non-repudiation impossible by keeping usage data.

Detectability ✓. To mitigate this threat, all communications are secured using TLS, notably the communication between the usage control system and the participants, the context handler and the PIPs, as well as the context handler and the PEPs. It also includes communications from the Distribution Management Point (DMP) to the DIFC center's Data Pre-processing Point (DPP), and communications between the DIFC center's components, i.e., the DPP, the DLP, and the PP.

Data disclosure ✓. The data processing is monitored by the usage control system, mitigating the excessive data volume threat. The usage control system, by monitoring both the information flow and the data usage, also prevents unnecessary processing. Excessive exposure is prevented as part of the access control to the data.

Table 5
LINDDUN privacy threat analysis, based on the illustrative scenario. ∼ if a threat is partially mitigated.

Threat	Scenario example	Mitigated by the UCS
Linkability	Link a car owner to a car renter	∼
Identifiability	Car owner is re-identified	∼
Non-repudiation	Car renter cannot deny car access	✗
Detectability	Detect a car owner is monitored by the UCS	✓
Data disclosure	Location data dissemination	✓
Unawareness	Participants not aware of location-sharing risks	✓
Non-compliance	Car renter policy not enforced	✓

Unawareness ✓. The usage control system requires car renters to design data policies, directly addressing the unawareness threat.

Non-compliance ✓. The usage control system monitors the car owners, stopping them from processing the location data unlawfully. Besides, the usage control system enforces the proper usage control policy specified by the car renter.

Usage control addresses four categories of threats directly, i.e., detectability, data disclosure, unawareness and non-compliance. However, linkability and identifiability threats are only partially mitigated. Only the non-repudiation threat is not addressed by usage control.

6. Conclusion

In this article, we propose an extended usage control model for distributed systems, by introducing elements of decentralized information flow control (DIFC) into the state-of-the-art formalism (Kelbert and Pretschner, 2018). The extension of the former modeling is justified by the need to consider all the aspects of the distribution, including the user-side policy definition, as well as specificities of IoT networks that require a more sophisticated distributed system modeling. Recent advances in DIFC, based on evolving coding practices (Liu et al., 2022), make DIFC usable and justify its consideration in distributed system modeling. Although this formalism enables the addition of external specification, using the dedicated evalExt operator, we state that DIFC actually requires dedicated components and functions. As a consequence, we extend the existing usage control architecture adding a privilege point (PP), a data pre-processing component (DPP) and a last component dedicated to data labeling (DL). Interfaces are also considered to enable cross-component communication.

Additionally, we define new functions to identify the relevant parts of the distributed system. The main function considered, conflict, detects individual systems $\mathcal{Y}$ (cf. Section 3.2) of the distributed system where the security policy on the data and their storage location are conflicting. Functions to handle the connection status of individual systems have also been introduced, serving a diversity of use cases where the network is unreliable or facing integrity threats, such as the Internet of Things or satellite networks. We have written a formal specification of our proposed model using the TLA+ language and we have used its attached model checker TLC to detect problems and subtle interactions before the implementation phase. We were able to detect potential loopholes due to the connection functions as well as temporal ordering issues and suggested mitigations. We also conducted a privacy analysis to highlight the usefulness of usage control.

CRediT authorship contribution statement

Nathanael Denis: Writing – original draft, Software, Formal analysis, Conceptualization. **Maryline Laurent:** Writing – review & editing, Supervision, Conceptualization. **Sophie Chabridon:** Writing – review & editing, Supervision, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The TLA+ specification is available in a public repository (https://doi.org/10.5281/zenodo.10721222).

Acknowledgments

This paper is supported by the Future and Ruptures program of Fondation Mines-Télécom, the Institut Mines-Télécom VP-IP Chair on Values and Policies of Personal Information (https://cvpip.wp.imt.fr) and the 3rd Programme d' Investissements d'Avenir (ANR-18-EUR-0006-02) within the framework of Energy4Climate Interdisciplinary Center (E4C) (https://www.e4c.ip-paris.fr/)

Appendix A. Modeling usage control policies with event-condition-action rule

ECA Rules. Event-Condition-Action (ECA) rules are expressed using the following operators:

$$\Psi \stackrel{def}{=} \underline{true} | \underline{false} | \mathcal{E}$$

$$\Omega \stackrel{def}{=} \underline{notIn}(\mathcal{D}, \mathcal{P}(C)) | \underline{combined}(\mathcal{D}, \mathcal{D}, \mathcal{P}(C)) | \underline{maxIn}(\mathcal{D}, \mathbb{N}, \mathcal{P}(C))$$

$$\Phi \stackrel{def}{=} (\Phi) | \Psi | \Omega | \underline{not}(\Phi) | \Phi \underline{and} \Phi | \Phi \underline{or} \Phi | \Phi \underline{since} \Phi | \Phi \underline{before} \mathbb{N} |$$

$$\underline{repmin}(\mathbb{N}, \mathbb{N}, \mathcal{E}) | \underline{repmax}(\mathbb{N}, \mathbb{N}, \mathcal{E}) | \underline{always}(\Phi) | \underline{evalExt}(\Gamma)$$

$$\Gamma \stackrel{def}{=} \Psi | \mathbb{N} | op | String | ...$$

In the above syntax, $\mathcal{D}$ represents a set of data items, $\mathcal{C}$ a set of containers and $\mathcal{E}$ a set of events. Γ is designed to incorporate external specifications and evaluation logic. It is left unspecified, but usually leverages XPath[4] for this purpose (Feth and Pretschner, 2012; Wüchner and Pretschner, 2012). This part of the formalism is meant to enable generalization. Operator $\underline{evalExt}$ allows to incorporate external specification and evaluation logics, to specify conditions that refer to subject and object attributes or environmental conditions, such as time and location. $\underline{evalExt}$ as a general operator can be used for DIFC incorporation but requires additional evaluation to ensure the incorporation's correctness (Kelbert and Pretschner, 2018).

Ω defines state-based operators that constrain the data flow state, which is directly related to the place of data within the containers. Notably, $\underline{notIn}(d, C)$ is true if the data d is not in any of the containers of C and $\underline{maxin}(d, m, C)$ is true if d is contained in at most m containers of C. In other words, Ω constrains the allowed states and by combination determines in which containers D must or must not be contained. The $\underline{combined}(d_1, d_2, C)$ operator is true if there exists at least one container $c \in C$ containing d_1 and d_2

[4] XPath (XML Path Language) is an expression language designed to support the query or transformation of XML documents. It enables the extraction of external specifications from XML documents.

Ψ intuitively refers to the constants true and false, but also to events $\mathcal{E}$. For instance, an event operator refine(e) where e is in $\mathcal{E}$ evaluates to true if and only if an event refining e is happening in the current timestep and the current data flow state. Finally, Φ defines the propositional, temporal and cardinality operators. not, and, or and always are intuitive. α since β is true if β was true at a moment earlier, i.e., during a former timestep and regardless of the current status, and α is true since then, or always true. α before $j \in \mathbb{N}$ is true if α was true exactly j timesteps ago. repmin and repmax correspond respectively to an event appearing at least or at most n times during the j last timesteps.

Appendix B. Formalism - Fundamentals

Identification functions To avoid unnecessary communication that would hinder performance, functions are needed to identify the relevant systems for evaluating the policy p. First, the three following auxiliary functions are defined (Kelbert and Pretschner, 2018):

(1) knowC: $\mathcal{P}(C) \rightarrow \mathcal{P}(\mathcal{Y})$. Given a set of containers C, knowC returns the set of systems in which one of the containers $c \in C$ resides. The formal definition of knowC is: $\forall C \in \mathcal{C} : \underline{knowC}(C) \stackrel{def}{=} \{y \in \mathcal{Y} | C_y \cap C \neq \emptyset\}$;

(2) knowD: $\mathcal{P}(\mathcal{D}) \times \mathbb{N} \times \prod \mathcal{T} \rightarrow \mathcal{P}(\mathcal{Y})$. Given a set of data items $\mathcal{D}$, a point in time $t \in \mathbb{N}$ and a tuple of concurrently executing traces $\mathcal{T}$, knowD relies on the systems' information flow to return the set of systems where one or more containers are storing at least one of the data items $d \in \mathcal{D}$:
$\forall D \subseteq \mathcal{D}, i \in \mathbb{N}, \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y : \underline{knowD}(D, i, \tau) \stackrel{def}{=} \{y \in \mathcal{Y} | \exists c \in \mathcal{C}_y, t_y^\tau \in \mathcal{T}_y, \sigma^i_{t_y^\tau} \in \Sigma_y : D \cap \sigma^i_{t_y^\tau}.s(c) \neq \emptyset\}$
knowD is parameterized in time and requires traces as input, as data are continuously propagated across systems;

(3) happens: $\mathcal{E} \times \mathbb{N} \times \prod \mathcal{T} \rightarrow \mathcal{P}(\mathcal{Y})$. Given an event $e \in \mathcal{E}$, a point in time, and a tuple of concurrently executing traces, happens returns the set of systems where any event refining e happens.
$\forall e \in \mathcal{E}, i \in \mathbb{N}, \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y : \underline{happens}(e, i, \tau) \stackrel{def}{=} \{y \in \mathcal{Y} | \exists t_y^\tau \in \mathcal{T}_y, e' \in t_y^\tau(i), \sigma^i_{t_y^\tau} \in \Sigma_y : (e', \sigma^i_{t_y^\tau}) \ \underline{refines}_\Sigma \ e\}$

Using these three auxiliary functions, the relevant function is composed so as to detect individual systems potentially interesting for the evaluation of policies. For a policy p, relevant: $\Phi \times \mathbb{N} \times \prod \mathcal{T}$ identifies the set of systems that may contribute to the evaluation of conditions $\varphi_p \in \Phi$ at a given timestep $i \in \mathbb{N}$ and given a set of concurrently executing traces $\tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y$. relevant is defined using the operators Ψ, Ω, Φ *and* Γ (cf. Fig. 7, Appendix A).

Operator relevant. The relevant function is defined to identify the relevant system sets for the evaluation of a usage control policy. It combines the logical operators Ψ, Ω, Φ and Γ with functions, i.e., knowC, knowD and happens (respectively introduced in Appendix A and Section 3.2). **Operator refines**. With the introduction of data flow states, it is possible to extend the refines operator to $refines_\Sigma$ which describes the refinement of an event in the presence of a given state. We say that (e_1, σ) refines e_2 if either:

(1) both e_1 and e_2 operate on the same container and if e_1 refines e_2;
(2) if e_1 operates on some container $c \in C$ and e_2 on some data $d \in D$ within c and e_1 refines e_2 when ignoring the parameter e.obj.

The formal definition of (e_1, σ) refines e_2 is :

$$\begin{aligned} &\forall e_1, e_2 \in \mathcal{E}, \sigma \in \Sigma : (e_1, \sigma) \ \text{refines}_\Sigma \ e_2 \\ &\stackrel{\text{def}}{\iff} \exists c \in \mathcal{C} : e_1.\text{obj} = c \wedge e_2.\text{obj} = c \wedge e_1 \ \text{refines} \ e_2 \\ &\vee \exists c \in \mathcal{C}, d \in \mathcal{D} : e_1.n = e_2.n \wedge e_1.\text{obj} = c \wedge e_2.\text{obj} = d \\ &\wedge d \in \sigma.s(c) \wedge e_{1..}.p \backslash \{(obj, c)\} \supseteq e_2.p \backslash \{(obj, d)\} \end{aligned} \quad (2)$$

Relating individual systems with system sets. We can define $\tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y$ as a set of traces of all individual systems. $t_y^\tau \in \mathcal{T}_y$ refers to the trace of the system $y \in \mathcal{Y}$ while σ_{t_y} refers to the corresponding data flow state. $t_Y^\tau \in \mathcal{T}_Y$ is defined as the combined traces of systems $Y \subseteq \mathcal{Y}$. For each timestep $i \in \mathbb{N}$, the set of events observed in Y, the union of the events in all individual systems $y \in Y$, such that:

$$\forall Y \subseteq \mathcal{Y}, i \in \mathbb{N}, \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, t_Y^\tau \in \mathcal{T}_Y : t_Y^\tau(i) \stackrel{def}{=} \bigcup_{y \in Y} t_y^\tau(i)$$

$\sigma_{t_Y^\tau}$ is constructed by unifying the mappings of the individual systems' storage, alias, and naming functions:

$$\forall Y \in \mathcal{Y}, i \in \mathbb{N}, \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, y \in Y, t_Y^\tau \in \mathcal{T}_Y, t_y^\tau \in \mathcal{T}_y, \sigma^i_{t_Y^\tau} \in \Sigma_Y, \sigma^i_{t_y^\tau} \in \Sigma_y,$$
$$c \in \mathcal{C}_y, j \in \mathcal{I}_y;$$

$$\Sigma_Y \subseteq \Sigma : (\mathcal{C}_Y \rightarrow \mathcal{P}(\mathcal{D})) \times (\mathcal{C}_Y \rightarrow \mathcal{P}_Y) \times (\mathcal{I}_Y \rightarrow \mathcal{C}_Y)$$

which leads to the definition of $\sigma^i_{t_Y^\tau}.s(c)$ such that:

$$\sigma^i_{t_Y^\tau}.s(c) \stackrel{def}{=} \sigma^i_{t_y^\tau} \wedge \sigma^i_{t_Y^\tau}.a(c) \stackrel{def}{=} \sigma^i_{t_y^\tau}.a(c) \wedge \sigma^i_{t_Y^\tau}.n(j) \stackrel{def}{=} \sigma^i_{t_y^\tau}.n(j)$$

Appendix C. Formalism - Extension

In this section, we detail the formalism of the newly introduced functions. First, the functions related to the use of DIFC are presented and then the functions associated with the connection status of individual systems.

C.1. Labeling functions

The formal definitions of the labeling functions introduced to handle DIFC aspects (cf. Section 4.1) are:

(1) labelC: $\mathcal{P}(C) \rightarrow \mathcal{P}(\mathcal{L})$. Given a set of containers, returns the labels associated with the containers: $\forall c \in C$, $\underline{labelC}(c) \stackrel{def}{=} \{l \in \mathcal{L} \ | c \in l.containers\}$;
(2) labelD: $\mathcal{P}(D) \rightarrow \mathcal{P}(\mathcal{L})$. Given a set of data, returns the labels associated with the data items: $\forall d \in \mathcal{D}$, $\underline{labelD}(d) \stackrel{def}{=} \{l \in \mathcal{L} \ | d \in l.data\}$
(3) conflict: $\mathcal{P}(C) \times \mathcal{P}(D) \rightarrow \mathcal{P}(\mathcal{Y})$. Given a set of containers C and data $\mathcal{D}$, returns the set of systems where there is a conflict between the containers and the data that a user is allowed to access. conflicts detects if there is at least one data item in d such that at least one reader r ($p \neq \emptyset$) is authorized to access d, but there is a container $c \in C_y$ knowing d which is forbidden to r. This function relies on knowD (cf. Appendix B) to identify the containers C knowing d in the first place: $\forall c \in \mathcal{C}, \forall d \in \mathcal{D}$, $\underline{conflict}(c, d) \stackrel{def}{=} \{y \in \underline{knowD(d)} \mid \exists l \in \mathcal{L}, l \in labelD(d) \wedge l \notin labelC(c)\}$;

C.2. Connection functions

Connection functions are introduced to handle the connection status in the network, necessary for a variety of use cases where the network is unreliable or if the integrity of individual systems is threatened (cf. Section 4.1). The formal definitions are:

(1) connected : $\mathcal{Y} \times \mathbb{N} \times \prod \mathcal{T} \rightarrow \mathcal{P}(\mathcal{Y})$. Given a set of individual systems $\mathcal{Y}$, a point in time $i \in \mathbb{N}$ and a tuple of traces, return the subset of individual systems that are reachable in the network at this point in time:

$$\forall y \in \mathcal{Y}, i \in \mathbb{N}, \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y, \underline{connected}(y, i, \tau) \stackrel{def}{=} \{y \in \mathcal{Y} \mid \forall e \in t_y^\tau(i), e.name \neq disconnect\}$$

$$\forall \varphi \in \Phi, i \in \mathbb{N}, \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y : \text{relevant}(\varphi, i, \tau) \stackrel{\text{def}}{=} \{y \in Y \mid$$

$$\left(\left(\varphi = \underline{true} \ \vee \varphi = \underline{false}\right) \Longrightarrow Y = \emptyset\right) \ \textbf{(1)}$$

$$\vee \exists e \in \mathcal{E} \cdot (\varphi = e \Longrightarrow Y = \text{ happens } (e, i, \tau)) \ \textbf{(2)}$$

$$\vee \exists d \in \mathcal{D}, m \in \mathbb{N}, C \subseteq \mathcal{C} \cdot ((\varphi = \underline{notIn}(d, C) \vee \varphi = \underline{maxin}(d, m, C)) \Longrightarrow \quad Y = \text{knowD}(\{d\}, i, \tau) \cap \text{knowC}(C)) \ \textbf{(3)}$$

$$\vee \exists d_1, d_2 \in \mathcal{D}, C \subseteq \mathcal{C} \cdot \left(\varphi = \underline{combined}\,(d_1, d_2, C) \Rightarrow Y = \text{knowD}\,(\{d_1\}, i, \tau) \cap \ \text{knowD}\ (\{d_2\}, i, \tau) \cap \text{knowC}(C)\right) \ \textbf{(4)}$$

$$\vee \exists \alpha \in \Phi \cdot (\varphi = \underline{not}(\alpha) \Longrightarrow Y = \text{relevant}(\alpha, i, \tau)) \ \textbf{(5)}$$

$$\vee \exists \alpha, \beta \in \Phi \cdot ((\varphi = \alpha \ \underline{and} \ \beta \vee \varphi = \alpha \ \underline{or} \ \beta) \Rightarrow \quad Y = \text{relevant}(\alpha, i, \tau) \cup \text{relevant}(\beta, i, \tau)) \ \textbf{(6)}$$

$$\vee \exists a, \beta \in \Phi \cdot (\varphi = \alpha \ \underline{since} \ \beta \Rightarrow \quad Y = \bigcup_{j=0}^{i} (\text{relevant}(\alpha, j, \tau) \cup \text{relevant}(\beta, j, \tau))) \ \textbf{(7)}$$

$$\vee \exists \alpha \in \Phi, j \in \mathbb{N} \cdot (\varphi = \alpha \ \underline{before} \ j \Rightarrow \quad Y = \text{relevant}(\alpha, i - j, \tau)) \ \textbf{(8)}$$

$$\vee \exists j, m \in \mathbb{N}, e \in \mathcal{E} \cdot (\varphi = \underline{repmin}(j, m, e) \Longrightarrow \quad Y = \bigcup_{k=0}^{\min(i,j)-1} \text{happens(e, } i - k, \tau))\} \ \textbf{(9)}$$

Fig. 7. Definition of the function relevant to identify systems of interest to evaluate a policy.

(2) availableC : $\mathcal{P}(C) \times \mathbb{N} \times \prod \mathcal{T} \rightarrow \mathcal{P}(\mathcal{Y})$. Given a set of containers, a point in time and a tuple of concurrently executing traces $\mathcal{T}$, availableC returns the set of currently reachable systems in which one of the containers $c \in C$ resides:

$$\forall C \in \mathcal{C}, i \in \mathbb{N}, \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y : \underline{availableC}(C)$$
$$\stackrel{def}{=} \{y \in \mathcal{Y} \mid C_y \cap C \neq \emptyset \ \wedge \underline{connected}(y, i, \tau) \cap \underline{knowC}(C) \neq \emptyset\};$$

(3) availableD : $\mathcal{P}(D) \times \mathbb{N} \times \prod \mathcal{T} \rightarrow \mathcal{P}(\mathcal{Y})$. Given a set of data items $\mathcal{D}$, a point in time $t \in \mathbb{N}$ and a tuple of concurrently executing traces $\mathcal{T}$, availableD returns the set of currently reachable systems where one or more containers are storing at least one of the data items $d \in \mathcal{D}$:

$$\forall D \subseteq \mathcal{D}, i \in \mathbb{N}, \tau \in \prod_{y \in \mathcal{Y}} \mathcal{T}_y : \underline{availableD}(D, i, \tau)$$
$$\stackrel{def}{=} \{y \in \mathcal{Y} \mid \exists c \in C_y, t_y^\tau \in \mathcal{T}_y, \sigma_{t_y^\tau}^i \in \Sigma_y : D \cap \sigma_{t_y^\tau}^i.s(c) \neq \emptyset$$
$$\wedge \underline{connected}(y, i, \tau) \cap \underline{knowD}(D, i, \tau) \neq \emptyset\}$$

References

Alvarez, Flor, Hollick, Matthias, Gardner-Stephen, Paul, 2016. Maintaining both availability and integrity of communications: Challenges and guidelines for data security and privacy during disasters and crises. In: 2016 IEEE Global Humanitarian Technology Conference. GHTC, pp. 62–70. http://dx.doi.org/10.1109/GHTC.2016.7857261.

Deng, Mina, Wuyts, Kim, Scandariato, Riccardo, Preneel, Bart, Joosen, Wouter, 2011. A privacy threat analysis framework: Supporting the elicitation and fulfillment of privacy requirements. Requir. Eng. 16 (1), 3–32. http://dx.doi.org/10.1007/s00766-010-0115-7.

Denning, Dorothy E., 1976. A lattice model of secure information flow. Commun. ACM 19 (5), 236–243. http://dx.doi.org/10.1145/360051.360056.

Ergun, Kazim, Ayoub, Raid, Mercati, Pietro, Rosing, Tajana, 2022. Reinforcement learning based reliability-aware routing in IoT networks. Ad Hoc Netw. 132, 102869. http://dx.doi.org/10.1016/J.ADHOC.2022.102869.

Fang, Tzu-Wei, Kubaryk, Adam, Goldstein, David, Li, Zhuxiao, Fuller-Rowell, Tim, Millward, George, Singer, Howard J., Steenburgh, Robert, Westerman, Solomon, Babcock, Erik, 2022. Space weather environment during the spacex starlink satellite loss in February 2022. Space Weather 20 (11), e2022SW003193. http://dx.doi.org/10.1029/2022SW003193, e2022SW003193 2022SW003193.

Feth, Denis, Pretschner, Alexander, 2012. Flexible data-driven security for android. In: Sixth International Conference on Software Security and Reliability, SERE 2012, Gaithersburg, Maryland, USA, 20-22 June 2012. IEEE, pp. 41–50. http://dx.doi.org/10.1109/SERE.2012.14.

Fromm, Alexander, 2020. Enhancing Data Flow Tracking for Data Usage Control (PhD Dissertation). Technische Universität München.

Gil, Gonzalo, Arnaiz, Aitor, Higuero, Marivi, Díez, Francisco Javier, 2022. Assessment framework for the identification and evaluation of main features for distributed usage control solutions. ACM Trans. Priv. Secur. 26 (1), 10:1–10:28. http://dx.doi.org/10.1145/3561511.

Harvan, M., Pretschner, A., 2009. State-based usage control enforcement with data flow tracking using system call interposition. In: 2009 Third International Conference on Network and System Security. pp. 373–380. http://dx.doi.org/10.1109/NSS.2009.51.

Kelbert, Florian, Pretschner, Alexander, 2013. Data usage control enforcement in distributed systems. In: Bertino, Elisa, Sandhu, Ravi S., Bauer, Lujo, Park, Jaehong (Eds.), Third ACM Conference on Data and Application Security and Privacy, CODASPY'13, San Antonio, TX, USA, February 18-20, 2013. ACM, pp. 71–82. http://dx.doi.org/10.1145/2435349.2435358.

Kelbert, Florian, Pretschner, Alexander, 2014. Decentralized distributed data usage control. In: Gritzalis, Dimitris, Kiayias, Aggelos, Askoxylakis, Ioannis G. (Eds.), Cryptology and Network Security - 13th International Conference, CANS 2014, Heraklion, Crete, Greece, October 22-24, 2014. Proceedings. In: Lecture Notes in Computer Science, Vol. 8813, Springer, pp. 353–369. http://dx.doi.org/10.1007/978-3-319-12280-9_23.

Kelbert, Florian, Pretschner, Alexander, 2015. A fully decentralized data usage control enforcement infrastructure. In: Malkin, Tal, Kolesnikov, Vladimir, Lewko, Allison Bishop, Polychronakis, Michalis (Eds.), Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers. In: Lecture Notes in Computer Science, Vol. 9092, Springer, pp. 409–430. http://dx.doi.org/10.1007/978-3-319-28166-7_20.

Kelbert, Florian, Pretschner, Alexander, 2018. Data usage control for distributed systems. ACM Trans. Priv. Secur. 21 (3), 12:1–12:32. http://dx.doi.org/10.1145/3183342.

Lamport, Leslie, 1992. Hybrid systems in TLA$^+$. In: Grossman, Robert L., Nerode, Anil, Ravn, Anders P., Rischel, Hans (Eds.), Hybrid Systems. In: Lecture Notes in Computer Science, Vol. 736, Springer, pp. 77–102. http://dx.doi.org/10.1007/3-540-57318-6_25.

Lamport, Leslie, 2002. Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley, http://research.microsoft.com/users/lamport/tla/book.html.

Li, Shancang, Xu, Li, Zhao, Shanshan, 2014. The internet of things: A survey. Inf. Syst. Front. 17, http://dx.doi.org/10.1007/s10796-014-9492-7.

Liu, Jason, Kandikuppa, Anant, Bates, Adam, 2022. Transparent DIFC: harnessing innate application event logging for fine-grained decentralized information flow control. In: 7th IEEE European Symposium on Security and Privacy, EuroS&P 2022, Genoa, Italy, June 6-10, 2022. IEEE, pp. 487–501. http://dx.doi.org/10.1109/EuroSP53844.2022.00037.

Lu, Jintian, Sun, Jiakun, Xiao, Ruizhi, Jin, Shuyuan, 2022. DIFCS: A secure cloud data sharing approach based on decentralized information flow control. Comput. Secur. 117, 102678. http://dx.doi.org/10.1016/j.cose.2022.102678.

Myers, Andrew C., Liskov, Barbara, 1997. A decentralized model for information flow control. In: Banâtre, Michel, Levy, Henry M., Waite, William M. (Eds.), Proceedings of the Sixteenth ACM Symposium on Operating System Principles, SOSP 1997, St. Malo, France, October 5-8, 1997. ACM, pp. 129–142. http://dx.doi.org/10.1145/268998.266669.

Owicki, Susan S., Lamport, Leslie, 1982. Proving liveness properties of concurrent programs. ACM Trans. Program. Lang. Syst. 4 (3), 455–495. http://dx.doi.org/10.1145/357172.357178.

Pretschner, Alexander, Lovat, Enrico, Büchler, Matthias, 2011. Representation-independent data usage control. In: García-Alfaro, Joaquín, Navarro-Arribas, Guillermo, Cuppens-Boulahia, Nora, di Vimercati, Sabrina De Capitani (Eds.), Data Privacy Management and Autonomous Spontaneus Security - 6th International Workshop, DPM 2011, and 4th International Workshop, SETOP 2011, Leuven, Belgium, September 15-16, 2011, Revised Selected Papers. In: Lecture Notes in Computer Science, Vol. 7122, Springer, pp. 122–140. http://dx.doi.org/10.1007/978-3-642-28879-1_9.

Rizos, Athanasios, Bastos, Daniel, Saracino, Andrea, Martinelli, Fabio, 2019. Distributed UCON in CoAP and MQTT protocols. In: ESORICS Int. Workshops, CyberICPS, SECPRE, SPOSE, and ADIoT, Luxembourg. In: LNCS, Vol. 11980, Springer, pp. 35–52. http://dx.doi.org/10.1007/978-3-030-42048-2_3.

Schultz, David, Liskov, Barbara, 2013. IFDB: Decentralized information flow control for databases. In: Proceedings of the 8th ACM European Conference on Computer Systems. EuroSys '13, Association for Computing Machinery, New York, NY, USA, pp. 43–56. http://dx.doi.org/10.1145/2465351.2465357.

Skarmeta, Antonio F., Hernández-Ramos, José L., Moreno, M. Victoria, 2014. A decentralized approach for security and privacy challenges in the Internet of Things. In: 2014 IEEE World Forum on Internet of Things. WF-IoT, pp. 67–72. http://dx.doi.org/10.1109/WF-IoT.2014.6803122.

The European Parliament and the Council of the European Union, 2018. General data protection regulation. http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679.

Wüchner, Tobias, Pretschner, Alexander, 2012. Data loss prevention based on data-driven usage control. In: 23rd IEEE International Symposium on Software Reliability Engineering, ISSRE 2012, Dallas, TX, USA, November 27-30, 2012. IEEE Computer Society, pp. 151–160. http://dx.doi.org/10.1109/ISSRE.2012.10.

Nathanael Denis (Ph.D.) graduated from Télécom SudParis, Institut Polytechnique de Paris, France, after conducting a Ph.D. under the supervision of Maryline Laurent and Sophie Chabridon. He is an Associated Researcher of the Chair of Institut Mines-Télécom "Values and Policies of Personal Information". His research interests include the Internet of Things, distributed systems, distributed ledgers, cryptography, usage control and privacy.

Maryline Laurent (Ph.D.) currently works as a Full Professor with the Samovar Laboratory, Télécom SudParis, Institut Polytechnique de Paris, France. She leads the RST Department (Telecommunication Networks and Services), Télécom SudParis, and she is also the Co-Founder of the Chair of Institut Mines-Télécom "Values and Policies of Personal Information. Her research interests include distributed ledger technologies, privacy enhancing technologies, and applied cryptography, the Internet of Things, cloud computing, and identity management systems.

Sophie Chabridon (Ph.D) works as a Professor and Director of Studies with the Samovar Laboratory, Télécom SudParis, Institut Polytechnique de Paris, France. She is also an Associated Researcher of the Chair of Institut Mines-Télécom "Values and Policies of Personal Information". Her research interests include the Internet of Things, privacy preservation, data consistency, databases, performance evaluation and ambient intelligence.