1 Privacy Levels nach Iteration 1

Privacy Level Ein Privacy Level sei eine wohlgeordnete Menge $(\mathcal{P}, \leq)$. Die Elemente $x \in \mathcal{P}$ können beliebig gestaltet werden, z.B. auch als n-Tupel. Es existiert auf $\mathcal{P}$ eine Halbordnung $\leq$ mit der Bedeutung:

$$\forall x, y \in \mathcal{P} : \; x \leq y :\Leftrightarrow \; y \text{ erlaubt gleich viel oder mehr als } x$$

$$\leftrightarrow x \text{ ist gleich restriktiv oder restriktiver als } y$$

(Wert-)Vorgabe Eine Vorgabe $s \in \mathcal{P}$ ist eine Mindestangabe eines Wertes. Erlaubt ein Wert $p \in \mathcal{P}$ gleich viel oder mehr als die Vorgabe, also $s \leq p$, so erfüllt dieser Wert die Vorgabe.

Korollar 1.1 *Betrachte s als Service-Level-Vorgabe. Dann kann das Service Level benutzt werden, wenn der Privacy Level $\mathcal{P}$ auf p eingestellt ist und $s \leq p$ gilt.*

Satz 1.2 *Es gebe n Presets, die jeweils die Werte $q_i \in \mathcal{P}$ haben. Dann wird eine Vorgabe s erfüllt, wenn für r aus Algorithmus 1 gilt: $s \leq r$.*

Algorithm 1 Algorithmus zur Preset-Kombination

```
r ← inf(P)
for i ← 1 ··· n do
  if r ≤ q_i then
    r ← q_i
  end if
end for
```

Anmerkung Es kann der Fall eintreten, dass $\mathcal{P}$ kein Infimum besitzt. In diesem Fall wird ein „virtuelles“ Infimum gewählt, also ein Element $z \notin \mathcal{P}$, für das willkürlich definiert wird: $\forall p \in \mathcal{P} : z \leq p$. z ist offensichtlich größte untere Schranke und somit Infimum.

Behauptung 1.3 *Algorithmus 1 berechnet in r das Maximum aller Presets. Mehr kann aus den Preset-Werten nicht erreicht werden. Gilt $s \leq r$, so folgt mit der Definition der Vorgabe, dass das Maximum die Service-Level-Vorgabe erfüllt. Damit ist Satz 1.2 bewiesen.*

Beweis Der Algorithmus setzt r zu Beginn auf den minimal möglichen Wert, also den Wert der am wenigsten erlaubt. Die Schleife, die darauf folgt, wählt jedes mal entweder den vorherigen Wert oder den größeren Preset-Wert, also denjenigen der mehr erlaubt. Nehme man an, q_m sei $\max\{q_1, \cdots, q_n\}$, aber $r \neq q_m$. Dann hat die If-Abfrage in Schleifen-Schritt m festgestellt, dass $r \not\leq q_m$, also $r > q_m$. Da r aber nur aus $\{q_1, \cdots, q_n\}$ ausgewählt wird, kann das nicht sein, denn q_m ist das Maximum dieser Menge. □

Anmerkung Die „Don't Care“-Einstellung von Presets muss so gewählt werden, dass sie als konkrete Konfiguration in q_i keine Auswirkung auf das Ergebnis von Algorithmus 1 hat. Es bietet sich offensichtlich entweder das Infimum an, falls es existiert, oder das Pseudo-Infimum z.

Beispiel (Brennweite) Die Ressourcengruppe Überwacherungskamera besitzt das Privacy-Level „Brennweite“ $\mathcal{F} := \{20, 40, 60, 80, 100\}$, mit der Schnappschüsse nur in bestimmten Zoomstufen gemacht werden dürfen. Die Halbordnung $\leq$ ist so wie auf den ganzen Zahlen definiert, größere Brennweiten erlauben mehr. Das aktive Preset „Allgemeine Überwachung“ definiert $q_1 = 40$, die Benutzereinstellung $q_2 = 60$ und ein anderes Preset steht auf „Don't care“, also $q_3 = 20$. Die App „Geheimüberwachung“ möchte für das Service-Level 1 „allgemeine Überwachung“ mindestens den Wert 40, also $s_1 = 40$; Service-Level 2 „detaillierte Überwachung“ benötigt mindestens den Wert 80, es ist also $s_2 = 80$.

PMP berechnet mit Algorithmus 1, dass $r = 60$. PMP vergleicht nun $s_2 \overset{?}{\leq} r$, also $80 \not\leq 60$. Service-Level 2 kommt also nicht in Frage. PMP vergleicht weiter $s_1 \overset{?}{\leq} r$, also $40 \leq 60$. Service Level 1 wird der App zugewiesen.

Beispiel (Datenbank-Lesen) Die Ressourcengruppe Datenbank besitzt das Privacy-Level „Lesen“ $\mathcal{R} := \{\bot, \top\}$, mit der festgelegt werden kann, ob die Datenbank gelesen werden darf. Die Halbordnung $\leq$ definiert $\bot \leq \top$, aber $\top \not\leq \bot$. Das aktive Preset „Datenbank-Standard“ definiert $q_1 = \top$, die Benutzereinstellung $q_2 = \bot$ und ein anderes Preset steht auf „Don't care“, also $q_3 = \bot$. Die App „Schach-Highscore“ möchte für das Service-Level 1 „Schach spielen“ keine Datenbank, also $s_1 = \bot$; Service-Level 2 „Highscoreanzeige“ benötigt mindestens Schreibzugriff, es ist also $s_2 = \top$.

PMP berechnet mit Algorithmus 1, dass $r = \top$. PMP vergleicht nun $s_2 \overset{?}{\leq} r$, also $\top \leq \top$. Service Level 2 wird der App zugewiesen.

Beispiel (Temperatur) Die Ressourcengruppe Backofen besitzt das Privacy-Level „Temperatur“ $\mathcal{T} := \{20, 150, 200, 250, \text{Grill}\}$, mit der die Backtemperatur der Funktionalität „Backen“ eingestellt werden kann. Die Halbordnung $\leq$ ist so wie auf den ganzen Zahlen definiert, heißere Temperaturen erlauben mehr, wobei Grill heißer ist als 250. Das aktive Preset „Kalter Ofen“ definiert $q_1 = 20$, die Benutzereinstellung $q_2 = 150$ und ein anderes Preset steht auf „Don't care“, also $q_3 = 20$. Die App „Pizza-Bäcker“ möchte für das Service-Level 1 „Tiefkühlpizza“ mindestens den Wert 200, also $s_1 = 40$; Service-Level 2 „Italienischer Steinofen“ benötigt mindestens den Wert Grill, es ist also $s_2 = \text{Grill}$.

PMP berechnet mit Algorithmus 1, dass $r = 150$. PMP vergleicht nun $s_2 \overset{?}{\leq} r$, also $\mathit{Grill} \not\leq 150$. Service-Level 2 kommt also nicht in Frage. PMP vergleicht weiter $s_1 \overset{?}{\leq} r$, also $200 \not\leq 150$. Service-Level 1 kommt also auch nicht in Frage. Service Level 0 wird der App zugewiesen.

2 Privacy Levels nach Vorschlag

(Mengen-)Vorgabe Eine Vorgabe ist eine Menge $S \subseteq \mathcal{P}$ an Werten. Erlaubt ein Wert $p \in \mathcal{P}$ gleich viel wie die Vorgabe, also ist $p \in S$, so erfüllt dieser Wert die Vorgabe.

Korollar 2.1 *Betrachte S als Service-Level-Vorgabe. Dann kann das Service Level benutzt werden, wenn der Privacy Level $\mathcal{P}$ auf p eingestellt ist und $p \in S$ gilt.*

Satz 2.2 *Es gebe n Presets, die jeweils die Werte $Q_i \subseteq \mathcal{P}$ haben. In den meisten Fällen ist $0 \leq |Q_i| \leq 1$, ein Preset definiert also einen oder keinen erlaubten Wert. Dann wird eine Vorgabe S erfüllt, wenn für R aus Algorithmus 2 gilt: $S \cap R \neq \emptyset$.*

Algorithm 2 Algorithmus zur Preset-Kombination

$R \leftarrow \emptyset$
for $i \leftarrow 1 \cdots n$ **do**
  $R \leftarrow R \cup Q_i$
end for

Beweis Algorithmus 2 berechnet in R offensichtlich die Vereinigung aller Presets. Mehr kann aus den Preset-Werten nicht erreicht werden. Gilt $S \cap R \neq \emptyset$, so folgt daraus, dass $\exists x \in \mathcal{P} : x \in R \Leftrightarrow x \in S$. Ein x, das sich in einem Preset befindet, erlaubt also gleich viel wie die Vorgabe S, da $x \in S$. Mit der Definition der Vorgabe ist diese erfüllt. Damit ist Satz 2.2 bewiesen.

Anmerkung Die „Don't Care"-Einstellung von Presets muss so gewählt werden, dass sie als konkrete Konfiguration in Q_i keine Auswirkung auf das Ergebnis von Algorithmus 2 hat. Es bietet sich offensichtlich die leere Menge $\emptyset$ an.

Beispiel (Brennweite) Die Ressourcengruppe Überwacherungskamera besitzt das Privacy-Level „Brennweite“ $\mathcal{F} := \{20, 40, 60, 80, 100\}$, mit der Schnappschüsse nur in bestimmten Zoomstufen gemacht werden dürfen. Das aktive Preset „Allgemeine Überwachung“ definiert $Q_1 = \{40\}$, die Benutzereinstellung $Q_2 = \{60\}$ und ein anderes Preset steht auf „Don't care“, also $Q_3 = \emptyset$. Der Benutzer hat darauf geachtet, dass nirgends 20 eingestellt wird, da die Totale der Kamera einige Dinge zeigt, die er nicht auf den Schnappschüssen zeigen will. Die App „Geheimüberwachung“ möchte für das Service-Level 1 „allgemeine Überwachung“ Werte zwischen 40 und 60, höhere Brennweiten sind unnötig. Also ist $S_1 = \{40, 60\}$; Service-Level 2 „detaillierte Überwachung“ benötigt mindestens den Wert 80, es ist also $S_2 = \{80, 100\}$.

PMP berechnet mit Algorithmus 2, dass $R = \{40, 60\}$. PMP stellt nun fest $S_2 \cap R = \emptyset$. Service-Level 2 kommt also nicht in Frage. PMP vergleicht weiter $S_1 \cap R = \{40, 60\} \neq \emptyset$. Service Level 1 wird der App zugewiesen.

Beispiel (Datenbank-Lesen) Die Ressourcengruppe Datenbank besitzt das Privacy-Level „Lesen“ $\mathcal{R} := \{\bot, \top\}$, mit der festgelegt werden kann, ob die Datenbank gelesen werden darf. Das aktive Preset „Datenbank-Standard“ definiert $Q_1 = \{\top\}$, die Benutzereinstellung $Q_2 = \{\bot\}$ und ein anderes Preset steht auf „Don't care“, also $Q_3 = \emptyset$. Die App „Schach-Highscore“ möchte für das Service-Level 1 „Schach spielen“ keine Datenbank, also $S_1 = \emptyset$; Service-Level 2 „Highscoreanzeige“ benötigt mindestens Schreibzugriff, es ist also $S_2 = \{\top\}$.

PMP berechnet mit Algorithmus 2, dass $R = \{\bot, \top\}$. PMP vergleicht nun $S_2 \cap R = \{\top\} \neq \emptyset$. Service Level 2 wird der App zugewiesen.

Beispiel (Temperatur) Die Ressourcengruppe Backofen besitzt das Privacy-Level „Temperatur“ $\mathcal{T} := \{20, 150, 200, 250, \text{Grill}\}$, mit der die Backtemperatur der Funktionalität „Backen“ eingestellt werden kann. Das aktive Preset „Kalter Ofen“ definiert $Q_1 = \{20\}$, die Benutzereinstellung $Q_2 = \{150\}$ und ein anderes Preset steht auf „Don't care“, also $Q_3 = \emptyset$. Die App „Pizza-Bäcker“ möchte für das Service-Level 1 „Tiefkühlpizza“ genau den Wert 200, also $S_1 = \{200\}$; Service-Level 2 „Italienischer Steinofen“ benötigt mindestens den Wert Grill, es ist also $S_2 = \{\text{Grill}\}$.

PMP berechnet mit Algorithmus 2, dass $R = \{20, 150\}$. PMP vergleicht nun $S_2 \cap R = \emptyset$. Service-Level 2 kommt also nicht in Frage. PMP vergleicht weiter $S_1 \cap R = \emptyset$. Service-Level 1 kommt also auch nicht in Frage. Service Level 0 wird der App zugewiesen.

3 Bericht

Die durchgeführte mathematische Untersuchung zeigt die theoretische Simplizität der Konzepte. Dennoch sind ihre Innovationsfolgen sowohl auf Anwender- als auch auf Entwicklerseite gravierend. Es gibt mindestens zwei weitere denkbare (und sinnvolle) Weiterentwicklungen des Vorschlags mit OR-Semantik:

1. Vorschlag mit AND-Semantik: Es müssen alle Werte gleichzeitig erfüllt sein (z.B. 40 und 60). Sinnvoll wäre dies für eine SQL-Datenbank mit einem Privacy Level „Befehle“ (SELECT, UPDATE, DELETE, ...). Die Forderung nach dem Algorithmus wird dazu auf $S \subseteq R$ verschärft, sodass alle Werte gleichzeitig erfüllt werden müssen.

2. Vorschlag mit OR-AND-Semantik: Es müssen alle Werte einer Vorgabenalternative gleichzeitig erfüllt werden. Die Vorgabe wird als Menge von Mengen definiert, sodass es sowohl Alternativen gibt, also auch Wertgruppen. $S_i := \{S_{i,1}, \cdots, S_{i,m}\}$. Die Forderung nach dem Algorithmus wird auf $\bigvee_{j=1}^{m} S_{i,j} \subseteq R$ erweitert. Ob eine Kombination sinnvoll ist, ist nicht abzusehen, allerdings kann dank der Methode sowohl AND- als auch OR-Semantik einzeln benutzt werden.

Um die Unterschiede zu verdeutlichen, habe ich eine Matrix angelegt:

	Iteration-1-PL	OR-PL	AND-PL	OR-AND-PL
Benötigte Operatoren	$\leq^1$, $\inf(\mathcal{P})$	$\leq^2$, $\cap$, $\cup$, $\overset{?}{=} \emptyset$	$\leq^2$, $\cup$, $\overset{?}{\subseteq}$	$\leq^2$, $\cup$, $\overset{?}{\subseteq}$
„Don't Care“	$\inf(\mathcal{P})$ oder z	$\emptyset$	$\emptyset$	$\emptyset$, als Vorgabe $\{\emptyset\}$
Preset-PL-Werte	1	beliebig	beliebig	beliebig
AND-Vorgaben	1	1	beliebig	beliebig
OR-Vorgaben	1	beliebig	1	beliebig
Datenstruktur	Wert	Menge	Menge	Menge von Mengen

[1] Diese Relation bedeutet „erlaubt mehr als“.

[2] Diese Relation wird nur zur Ordnung der Werte benötigt.

Es zeigt sich eindeutig, dass Privacy-Level nach der 1. Iteration die simpelste Struktur sind, die auch einfach zu implementieren ist. Der Privacy-Level-Entwickler muss lediglich die Ordnung $\leq$ definieren. Es werden höchstens so viele Nachrichten an das Privacy Level geschickt, Werte zu vergleichen, wie es Service Level gibt.

Allerdings zeigt sich auch, dass sie vielen Begrenzungen unterliegen. Das mathematische Modell ist nicht so gut definierbar, z.B. wegen z. Für Privacy-Level auf denen es keine offensichtliche $\leq$-Relation mit der Bedeutung „erlaubt mehr als“ bzw. „ist besser als“ gibt, (z.B. Versionen, Befehlssätze oder physikalische Werte mit einem Optimum $\neq 0$) ist die Methodik gar fragwürdig. Als Service-Level-Vorgabe sind Aussagen der Form „Mir würde schon reichen, wenn...“, „Ich brauche nur ... und ...“, „Entweder ... oder ...“ nicht möglich, insofern sie nicht als eigene Privacy-Levels definiert sind. Ob der Ressourcen-Entwickler diese Weitsicht hat, ist zu bezweifeln, ebenso, dass die bereitgestellten Möglichkeiten ausreichen würden.

Gegen die vorgeschlagenen Privacy-Level-Implementierungen spricht ganz klar ihr Aufwand, sowohl für die PMP-Entwickler, als auch für die Ressourcen-Entwickler. Es muss eine eigenständige, in der Datenbank serialisierbare Mengenklasse entwickelt werden, die entweder nur über eine Ordnung $\leq$ (ohne semantische Information) Mengen-Operationen bewältigen kann. Dann wird die Menge dabei allerdings viele Anfragen an das Privacy-Level stellen. Die Alternative hierzu wäre die Ordnung $\leq$ fallen zu lassen und im Privacy-Level die

Mengen-Operationen definieren zu lassen. Dies ist selbstverständlich komplexer zu programmieren als eine Halbordnung. Mengen mit unendlicher Kardinalität sind zu berücksichtigen, sollten sich aber analog verhalten. Die Darstellung und Einstellung dieser Mengen in Presets und Service-Levels dürfte weiterhin zu Problemen führen. Zudem wird die Konzeption von Privacy-Levels und Definition von Service-Levels umständlicher (z.B. wenn der Entwickler versehentlich „false“ statt $\emptyset$ fordert). Die Speicherung und Berechnung der Daten dürfte komplizierter sein als nur einen Wert abzulegen. Die Notwendigkeit für die AND-OR-Semantik sollte bei den meisten (aber längst nicht bei allen) Privacy Levels nicht bestehen.

Allerdings sind auch die Vorteile der Vorschläge klar: Der Wegfall der $\leq$-Relation ermöglicht ein breiteres Anwendungsspektrum, da die Werte nicht mehr willkürlich in eine „besser-als“-Halbordnung gezwängt werden müssen. Gleichzeitig ermöglichen sich durch die AND-OR-Struktur Aussagen in disjunktiver Normalform, es sind also (theoretisch) alle Kombinationen von AND- und OR-Verknüpfungen im optimalen Fall möglich. Obwohl Missverständnisse bei der Service-Level-Definition möglich sind („false“ statt $\emptyset$) vermeidet sie gleichzeitig auch mögliche Missverständnisse. So ist bei den Privacy-Levels nach der 1. Iteration wohl möglich, dass es Privacy-Levels gibt, die zwar Boolean sind, aber „false“ besser als „true“ einschätzen, z.B. wenn sie in Form einer Verneinung formuliert werden. Im vorgeschlagenen Ansatz muss hier zwar mehr nachgedacht werden, es entsteht aber auch eine eindeutigere Lösung. Während bei Privacy-Level-Werten, die ja nur zum Zugriff auf Ressourcen und Funktionalität benötigt werden, diese Vorteile noch sehr nach Minderheitsfällen klingen, so scheinen sie bei Kontexten wesentlich häufiger vorzukommen (die meisten spontan erdachten Beispiele sind Kontext-Beispiele). Aus Konsistenzgründen würde sich eine Vereinheitlichung anbieten, zumal womöglich für Kontexte sowieso die Datenstrukturen entwickelt werden müssen.

Abschließend lässt sich sagen, dass sich gezeigt hat, dass die vorgeschlagene Verfeinerung der Privacy-Levels nicht nur Vorteile in den Punkten Flexibilität, Anwender-Benutzbarkeit und Verständlichkeit aufweist, sondern auch Nachteile in den Punkten Komplexität, Entwickler-Benutzbarkeit und Verständlichkeit mit sich bringt. Es ist daher nicht möglich, eine konkrete Variante als beste hervorzuheben, sondern stattdessen nötig, genau abzuwägen, welche Variante letztendlich gewählt wird.