Compiling Personal Data and Subject Categories from App Data Models

Christian Burkert, Maximilian Blochberger, Hannes Federrath

Compiling Personal Data and Subject Categories from App Data Models

Christian Burkert, Maximilian Blochberger, and Hannes Federrath

University of Hamburg
{burkert,blochberger,federrath}@informatik.uni-hamburg.de

Abstract Maintaining documentation about personal data processing is mandated by GDPR. When it comes to application software and its operation, this obligation can become challenging. Operators often do not know enough about app internals to be comprehensive in their documentation or follow changes enough to be up-to-date. We therefore propose a semi-automatic process to compile documentation from the source of truth: the app data model. Our approach uses data model entity relations to determine identifiability of data subjects. We guide app experts to add the semantic knowledge that is necessary to determine subject categories and to subsequently compile a condensed listing of personal data. We provide evidence for the real-world applicability of our proposal by evaluating the data models of five common web apps.

Keywords: Data protection · Personal data identification · Data model

1 Introduction

GDPR requires operators of application software to create and maintain documentation about their processing of Personal Data (PD)[1], as stated in Article 30. Documenting such processing activities includes 'a description of the categories of data subjects and of the categories of personal data' (Article 30 (1)(e)). This requires the compilation of categorical descriptions for PD and semantic subject roles that are sufficiently detailed to, amongst others, allow regulatory authorities to assess the proportionality of data processing. However, to keep such documentation complete, correct, and up-to-date, requires great care and ongoing observation of changes to the software. Without automation, this will likely result in errors or deviation from the app as the source of truth. We therefore propose *Schemalyser*, a semi-automatic process to derive PD and subject categories from app data models. The underlying data models in the form of entities, attributes of entities and relations between entities can be extracted from various sources like source code [5] or database (DB) schemes [1] created by the app. Thereby, we are not reliant on the usage of relational DB. Schemalyser assumes that data subjects, i. e., the identifiable natural persons, are themselves

[1] We do not use the abbreviation PII to avoid confusion with non-GDPR definitions.

represented as one or more entities in the data model, which is usually the case for (collaborative) web apps and many other client-server software. To determine which data is personal and thus needs to be included in the compliance documentation, we can therefore utilise entity relations and the information whether data is connected to a data subject entity and how. This is in accordance with Article 4(1) GDPR, which defines that, what makes personal data personal, i. e., relatable to a natural person, is less a property of the data itself but more of its semantic context and the identifiability within.

By analysing five real-world app data models, we observed a high degree of interconnectivity among entities: Entities representing data subject are directly connected with 40 to 70 % of all entities and over 80 % are indirectly reachable. In other words, data subjects are identifiable for almost all application data. As a result, the documentation for PD of an app would comprise hundreds or thousands of attributes, thus loosing its informative purpose. Moreover, a user's name is indistinguishable in terms of identifiability from the number of likes the user has received for a chat posting, which raises the question of prioritized presentation. To the best of our knowledge, we are the first to tackle this *ubiquitous identifiability* problem by introducing differentiated identifiability classes and provide a process to compile condensed PD listings per subject category.

Our main contributions are: *a*) we describe the problem of ubiquitous identifiability in data models, *b*) we propose a semi-automatic expert process to derive PD and subject categories directly from the source of truth, and *c*) we provide evidence for the complexity of real-world data models and practical applicability.

The remainder of this paper is structured as follows: We first present related work in Sect. 2 before we introduce the Schemalyser approach in Sect. 3. Section 4 evaluates our proposal. Finally, we discuss the integration in development workflows in Sect. 5 and conclude in Sect. 6.

2 Related Work

As far as we are aware, we are the first to propose a process to semi-automatically compile Article 30-related compliance documentation from app data models. Martin and Kung [7] envision a data model-driven process to inventory personal data but do not name any existing approaches. Fakas et al. [4] propose a similar mechanism but to semi-automatically create responses to subject access requests, whereby data entries are identified by a human keyword search. Then, starting from matching instance, the operator iteratively browses neighbouring relations, and selects entities and attributes for the response. In contrast to Schemalyser, they do not consider identifiability classes, relation-defined roles or compiling categories. Furthermore, Schemalyser analyses abstract data models and derives information about instances that can potentially exist.

A similar approach, is used in [6] to generate Record of Processing Activitiess (RPAs) from formal Enterprise Architecture models and derive data recipients based on relations between business processes. However, individual business processes and their categories of PD and subjects are documented through expert

interviews and out of scope in terms of composition. Regarding RPA best practices especially concerning the categories of PD and subjects, a recent analysis of RPA templates in [8] did not include categorisation aspects into their semantic model. Bercic and George [2] discuss identifiability in DBs from a legal pre-GDPR view and also conclude that all data that is linkable to a subject within a DB should be considered PD.

3 Schemalyser Approach

Our approach builds on a graph representation of the data model where the vertexes represent the data model entities and the edges represent directed 1:N relations between entities, i. e., foreign key (FK) relations. Since there can be multiple FK relations between the same two entities, the result is a multigraph. We call this graph a *scheme*: A scheme S is defined as a multigraph $S = (E, R)$ with a set of entities E and FK relations $R \subseteq E \times E \times \mathbb{N}$, where $(e_1, e_2, i) \in R$ denotes the i-th FK from entity e_1 to entity e_2. Note that entity attributes are left out for simplicity at this stage. For legibility, we will use the notation $(e_1, e_2) \in R$ to express that there exists an $i \in \mathbb{N}$ such that $(e_1, e_2, i) \in R$.

Based on this graph representation, our approach takes four partially automated steps to derive a condensed listing of PD per subject category, which are described in detail in the following.

3.1 Seed Identification

To determine what data within the scheme is potential PD, we first need to identify one or more *seed* entities: A seed $s \in E$ is an entity in the data model that represents a data subject or a role (e. g., user, reporter) and is modelled as a dedicated entity. A scheme can contain multiple seeds. Correctly and completely identifying all seeds requires domain knowledge about the app data model. However, we find that seeds typically stand out as central entities within the data model (cf. Sect. 4.1). Thus, to speed up their identification, we propose to rank the entities according to their degree centrality.

3.2 Identifiability Markup

Given the seeds as representations of data subjects in the data model, the second step now determines for the remaining entities a notion of identifiability with respect to each seed. In other words: does the data model associate a given entity with one or more of the seeds and if so how.

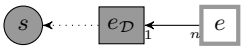
The naive approach would be to ignore the direction of relations and calculate the reachability graph of each seed s in an undirected scheme. We call this the partition $P_s \subseteq E$, which is defined recursively from base $\{s\}$ as:

$$P_s = \{s\} \cup \{e_2 \in E \mid \exists e_1 \in P_s \colon (e_1, e_2) \in R \vee (e_2, e_1) \in R\}$$

However, this simplistic approach neglects the cardinalities implied by the directionality of the relations, i. e., the direction implies whether an entity is related to exactly one instance of another entity or an arbitrary number of such instances. This distinction determines whether a relation to a seed allows to single out an individual data subject or a potential multitude of data subjects. Following this distinction, we subsequently formalise the direction semantic and introduce classes of identifiability.

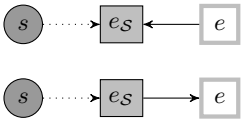
Relation Direction Semantics We distinguish two semantic sub-relations between a pair of entities depending on the direction of the FK relation: *extending* and *providing*:

1. *Extending* $e_1 \to e_2$ (n-to-1): Each instance of e_1 extends the context of exactly one instance of e_2 and is unambiguously associated with that instance.
2. *Providing* $e_1 \leftarrow e_2$ (1-to-n): An instance of e_1 can provide non-exclusive context to an arbitrary number of instances of e_2.

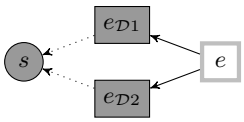


(a) Dedicated: n instances of entity e are related to a single instance of $e_\mathcal{D} \in \mathcal{D}_s$.

(b) Shared 'Original': An instance of e provides context for n instances of $e_\mathcal{D} \in \mathcal{D}_s$.



(c) Shared 'Extended': Entity e further extends or provides to shared entity $e_\mathcal{S} \in \mathcal{S}_s$.



(d) Shared 'Merged': Entity e is shared by extending s via multiple $e_{\mathcal{D}1}, e_{\mathcal{D}2}, \ldots \in \mathcal{D}_s$.

Figure 1. Identifiability classes for a seed from the perspective of other entities.

Identifiability Classes Based on these relation semantics, we identified three class of identifiability of a seed s regarding another entity. The classes are not necessarily exclusive and are recursively defined as subsets of E as follows:

1. *Dedicated*: An entity is dedicated to a seed if it has a direct, extending FK path to the seed (see Fig. 1a):

$$\mathcal{D}_s = \{s\} \cup \{e \in E \mid \exists e_\mathcal{D} \in \mathcal{D}_s : (e, e_\mathcal{D}) \in R\}$$

   Example: A user (seed) has multiple addresses, e. g., invoice and delivery.
2. *Shared*: An entity has one or more indirect paths to the seed by providing for a dedicated entity (*original share*), by extending or providing for another

shared entity (*extended share*), or by extending dedicated entities through two or more distinct FK paths (*merged share*), see also Figs. 1b to 1d:

$$\begin{aligned}
\mathcal{S}_s &= \mathcal{S}_s^{Orig} \cup \mathcal{S}_s^{Ext} \cup \mathcal{S}_s^{Merged} \\
\mathcal{S}_s^{Orig} &= \{e \in E \mid \exists e_{\mathcal{D}} \in \mathcal{D}_s \colon (e_{\mathcal{D}}, e) \in R \\
&\qquad \wedge (e_{\mathcal{D}} = s \vee \exists e_{\mathcal{D}'} \in \mathcal{D}_s, e_{\mathcal{D}'} \neq e \colon (e_{\mathcal{D}}, e_{\mathcal{D}'}) \in R)\} \\
\mathcal{S}_s^{Ext} &= \{e \in E \mid \exists e_{\mathcal{S}} \in \mathcal{S}_s \colon ((e, e_{\mathcal{S}}) \in R \vee (e_{\mathcal{S}}, e) \in R) \\
&\qquad \wedge (\exists e_{\mathcal{D}} \in \mathcal{D}_s, e_{\mathcal{D}} \neq e \colon (e_{\mathcal{D}}, e_{\mathcal{S}}) \in R \\
&\qquad \vee \exists e_{\mathcal{S}'} \in \mathcal{S}_s, e_{\mathcal{S}'} \neq e \colon ((e_{\mathcal{S}'}, e_{\mathcal{S}}) \in R \vee (e_{\mathcal{S}}, e_{\mathcal{S}'}) \in R))\} \\
\mathcal{S}_s^{Merged} &= \{e \in E \mid \exists e_{\mathcal{D}1}, e_{\mathcal{D}2} \in \mathcal{D}_s \colon \{(e, e_{\mathcal{D}1}), (e, e_{\mathcal{D}2})\} \subseteq R\}
\end{aligned}$$

In $\mathcal{S}_s^{Orig}$ and $\mathcal{S}_s^{Ext}$ constraints are necessary to avoid that a *shared* classification propagates backwards to entities that caused it. In Fig. 1b, for instance, $e_{\mathcal{D}}$ does not receive the extended shared membership from e, because $e_{\mathcal{D}}$ is the sole origin of e's shared membership. Also note that the merged share classification is assigned after all entities have been otherwise classified and the dedicated, shared original and shared extended sets are stable. It does not cause further extended shares.

Example: Multiple users are participants in a chat room. Information about the chat room is attributable to all participants without being able to single out one participant.

3. *Unrelated*: There is no relation between the seed and the other entity:

$$\mathcal{U}_s = E \setminus P_s$$

Example: Global configuration of the application.

Based on these three identifiability classes, we can distinguish whether an attribute, according to the entity it belongs to, solely concerns an individual data subject (dedicated), potentially relates to a set of data subjects (shared), or can most likely be excluded from consideration as PD (unrelated).

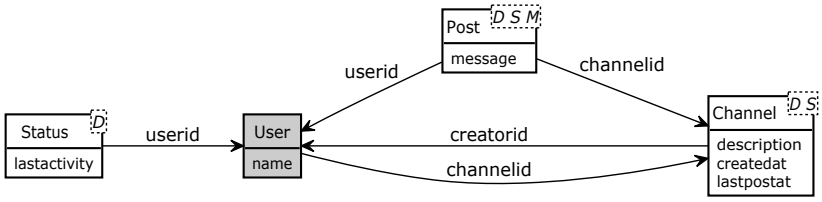


Figure 2. Simple chat app data model with identifiability classes (D)edicated, (S)hared, and shared (M)erged assigned to each data model entity.

Example To illustrate the classification, Fig. 2 shows the resulting identifiability classes for a data model of a simple chat app. In this example, the User entity is the seed. The other entities all have at least one dedicated path to User and therefore receive the `D` label. The relation between User and Channel through channelid is providing, hence Channel receives the original shared label `S`, which is then carried over as extending shared to Post. And finally, as Post has two dedicated paths, one direct and one via Channel, it receives the merged shared label `M` which implies `S`, because each dedicated path might associate a different user to Post that way, all of which then share Post's attributes.

3.3 Role Determination

During this step, a domain expert with knowledge about the data model is guided to determine roles of a user that are reflected in the data model. These roles eventually build the basis for categorising data subjects and grouping PD. We regard a role as a semantic distinction of a user's relation to the system that is characterised by the user engaging with the system in an optional manner. By being optional, it differentiates itself from a general engagement with the system, i. e., the default user role. For instance, a user chooses to report an issue by which they take the role of an issue reporter. In the data model, roles can be modelled in different ways, e. g., by having dedicated role entities or explicit FK attributes for a role. We find that roles are typically defined by the entities and attributes directly surrounding the seed, i. e., the first hops on a path between the seed and other entities. We call those *first-hop relations*. However, not every first-hop relation necessarily constitutes a role, because it might lack a defining characteristic like the optionality of the engagement. This is a semantic distinction that cannot generally be inferred solely based on the scheme. Therefore, during this step of determining roles, a domain expert has to classify, which first-hop relations define a role. For this classification process, we propose the following classes:

- *Integral*: The relation adds data for users in general that is integral and unconditional to the overall system functionality.
- *Role*: The relation describes an optional user engagement with the system that defines their capabilities and perception by the system and other users.
- *Conditional*: The relation provides details related to the usage of an optional feature which, other than a role, does not define their perception because it is less functionally significant or visible.

The distinction between roles and conditionals is gradual. Relations like the uploader of a file attachment to a post could be regarded as both a uploader role or a conditional usage of the feature to attach files to a post. We argue that, in this example, uploader should be considered a conditional because it is less significant and visible than the poster role which coincides with uploading a file. As roles will later form subject categories that should be meaningful to non-experts, they should be defined sparingly.

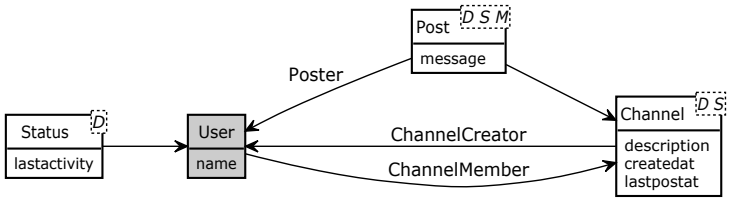


Figure 3. Minimal chat app data model with three determined roles.

Revisiting the chat app example in Fig. 3, we deemed three out of the four first-hop relations as role-defining. The fourth relation between Status and User is obligatory for each user and thus integral.

3.4 Decisive Role Selection

During this step, a domain expert selects for each attribute of every entity with a dedicated and/or shared classification, which of the previously determined roles are decisive for this attribute. We consider a role as *decisive* for an attribute if the attribute's value is derived from a property or action of a member of that role. Potential candidates for decisive roles are all roles whose defining relation lies on a path from the respective entity to the seed. Note, that as we are only considering simple paths, i. e., paths where each node is at most visited once, there is always only one first-hop relation and thus only one role-defining relation on each path. We call the candidates that are deemed as non-decisive *subjected roles*. Subjected roles might be equally identifiable than the decisive role of an attribute. The distinction rather allows to prioritise and filter attributes as will be shown in the final step. For instance, a channel creator is subjected by messages of posts in their channel but does not decide them. In the following, we describe the substeps of enumerating paths and selecting candidates as well as lower-complexity alternatives for large data models.

Path Enumeration To determine the candidates for decisive roles for a given seed and entity, we first have to find all paths between that seed and entity. However, we cannot simply list all simple paths between these nodes, because the paths have to account for the relation direction semantics described in Sect. 3.2 and the resulting propagation of identifiability classes. We denote the paths through which entities received their dedicated or shared classification as dedicated and shared paths, respectively.

As the propagation of the dedicated class follows only extending edges that are directed towards the seed, dedicated paths are strictly directed walks in the graph. Hence, dedicated paths are simple paths in a directed scheme graph with inverted directions. Regarding shared paths, we have to consider graph walks with mixed directions. However, as defined in Sect. 3.2, a shared path contains at least one providing relation and allows no back-propagation. Shared paths are

Table 1. Enumeration of dedicated and shared paths for the chat app example.

Entity	$\mathcal{D}$ Paths	$\mathcal{S}$ Paths
Status	$\{U \leftarrow S\}$	$\{\}$
Post	$\{U \leftarrow P, U \leftarrow C \leftarrow P\}$	$\{U \rightarrow C \leftarrow P\}$
Channel	$\{U \leftarrow C\}$	$\{U \rightarrow C, U \leftarrow P \rightarrow C\}$

therefore a subset of paths in an undirected graph. To illustrate this, Table 1 enumerates the dedicated and shared paths of the chat app from Fig. 4. Note that in this example, ignoring the constraints for shared paths would for instance incorrectly add the shared paths $U \leftarrow P$ and $U \leftarrow C \leftarrow P$ for entity Post, which would lead to a enlarged and misleading PD listing for the roles Poster and Channel Creator in the final step.

Candidate Selection Candidates for a given entity are all roles whose defining relation lies on of the previously enumerated dedicated or shared paths. For large data models, the number of candidates might still be quite high. To speed up the manual selection process by the domain expert, we propose to present the candidates in descending order according to the following selection likelihood heuristic:

1. dedicated-only roles before mixed before shared-only roles, then
2. roles with shortest paths first, then
3. roles with lowest number of paths first.

We argue that dedicated-only roles are more likely decisive than mixed and shared-only roles because an identifiable individual is more likely the originator of an action or actively involved than a set of individuals. Regarding the influence of path length, we argue that shorter paths imply a more direct logical connection. The third ordering rule presumes that a higher number of paths using the same first-hop relation decreases the semantic specificity and is thus more likely to be a modelling artefact than semantically significant.

In completing this step, the domain expert selects one or more decisive roles for each attribute. Figure 4 shows the decisive role selection for the attributes of the chat app example. The Poster role is selected as decisive for the message of a post. The attributes of Channel have each a different decisive role. This selection is modelled after the Mattermost chat app, where the ability to set and change the description of a channel is given to every member of that channel.

Complexity and Path Enumeration Alternative Listing all paths between the seed and the other entities has a $O(|E|!)$ computational and spacial complexity. For large and highly interconnected data models, it might therefore be practically infeasible to use the path enumeration approach, as we will show in Sect. 4. For those cases, it might be a sufficient compromise to determine candidates by checking for reachability via each role-defining relation instead. To do

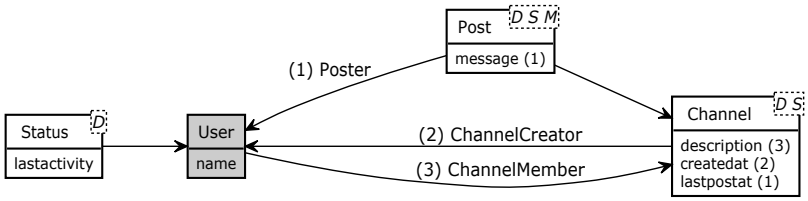


Figure 4. Minimal chat app data model with decisive roles selected.

so, we check for each role-defining relation which entities are still reachable from the seed if all but this first-hop relation were removed. Given that the complexity of a reachability check between a single pair of entities is $O(|E| + |R|)$, the complexity for checking the reachability of a seed and all other entities via each role-defining relation is therefore $O(r|E|(|E| + |R|))$, where r is the number of role-defining relations. In the worst case of a fully meshed graph where $r = |E|$ and $O(|R|) = O(|E|^2)$ the complexity becomes quadratic. Despite reachability checking being a more efficient alternatives for large inputs, we recommend the path enumeration where possible, because being able to inspect a path can provide valuable context information to the domain expert.

3.5 Condensed PD Listing

For this final step, we propose a condensed per-role listing of PD to maintain readability and focus on those attributes that have a higher significance for a given role. To achieve this condensed listing, we use the dedicated/shared distinction and the decisive role selections to determine if an attribute should be listed as PD for a given role in detail or if it can be aggregated into a *grouping term*. To list PD for a given role, we list an attribute explicitly if the role is decisive for that attribute. If not, the attribute is represented by a grouping term. As a result, we have four sets of PD for each role that decrease in identifiability and significance:

1. *Dedicated decisive*: The attribute is significantly determined by a member of this role and the member can be singled out.
2. *Dedicated non-decisive*: The attribute is associated with an identifiable member of this role without being determined by them.
3. *Shared decisive*: The attribute is significantly determined by a member of this role but *no* member can be singled out.
4. *Shared non-decisive*: The attribute is not determined by this role and *no* member can be singled out.

The grouping terms used in the non-decisive sets can represent a subset of attributes of an entity, an entity as a whole, or multiple semantically related entities. A display name of an entity, i. e., a name dedicated for presentation to users, could for instance be used as an automatically derived term. In cases where such derived grouping terms might be too incomprehensible for the general public,

a domain expert should assign fitting grouping terms manually. If the media supports hypertext, grouping terms should be expandable to the individual attributes summarised by them.

To illustrate the condensed listing, we again use the data model and roles from the chat app as shown in Fig. 4. The resulting PD listing is shown in Table 2. Entity names are used as grouping terms.

Table 2. Condensed PD listing for the chat app. Cursive entries are group terms.

Role	1. $\mathcal{D}$ decisive	2. $\mathcal{D}$ non-dec	3. $\mathcal{S}$ decisive	4. $\mathcal{S}$ non-dec
User	status lastactivity	-	-	-
Poster	message lastactivity	-	-	*Channel data*
Channel Creator	createdat	*Channel data* *Post data*	-	-
Channel Member	-	-	description	*Channel data* *Post data*

4 Evaluation

In this section, we evaluate the practicality of our proposal in terms of number and difficulty of necessary interaction, the complexity, as well as the degree of condensation achieved in the PD listing. To also gain an insight into costs for real-world apps, we used five common web apps (see Table 4) and analysed their data models. Lacking the necessary domain expertise for the software architecture of all those apps, we were only able to evaluate the manual steps for the Mattermost app, which we have extensively studied before [3].

4.1 Interaction Cost and Complexity

In this section, we assess the number of manual actions and the number of alternatives a domain expert has to choose from during each process step. An overview of the costs and complexities is provided in Table 3.

Seed Identification This requires one manual step to select all seeds from a list of all entities. However, we find that seeds typically exhibit a high degree centrality and rank in the top two entities with the highest degree centrality in the tested apps, ranging from 41 to 69 %. Therefore, the expert does not need to inspect all entities at this point. Calculating the degree centrality over the graph adds a $O(|E|^2)$ complexity. Note that in the following, we will assume that the number of seeds in a data model does not grow with the number of

Table 3. Overview of the cost and complexity of our proposal.

Step	#Actions	#Decisions	Worst Case Complexity
1. Seed Identification	1	$O(1)$	$O(\|E\|^2)$
2. Identifiability Markup	-	-	$O(\|E\|^2(\|E\|+\|R\|)\deg_{\max})$
3. Role Determination	$O(\|E\|)$	3	-
4. Decisive Role Sel.	$O(\|E\|)$	$O(\|E\|)$	$O(\|E\|!)/O(\|E\|^4)$
5. PD Listing	$O(\|E\|)$	1	$O(\|E\|^2)$

entities. Instead, we argue that the number of seeds depends on modelling styles and is typically in the low single digits. Our evaluation examples support this assumption having each only a single seed.

Identifiability Markup Assigning identifiability classes is a fully automatic process. Our proof-of-concept implementation uses a breadth-first (BFS) approach to propagate the classes through the scheme graph. It repeats until the classes are stable, i. e., until it has gathered for each entity and identifiability class, all neighbouring entities that propagate that label to that entity. Note, that gathering all propagation origins is necessary to avoid back propagation (cf. Sect. 3.2). During a single BFS traversal we process each entity's neighbours, which leads to a complexity of $O((|E|+|R|)\deg_{\max})$ complexity, where $\deg_{\max}$ is the maximum node degree. To reach stability, a theoretical worst case of $|E|^2$ repeats would be necessary if every entity receives its classes from every other entity but with only a single new propagation per iteration. However, this is a very conservative estimation, since an increase in connectivity would also speed up propagation. In practice, reaching stability took 4 to 5 iterations for our test apps. In total, this step has at worst a $O(|E|^2(|E|+|R|)\deg_{\max})$ complexity.

Role Determination During this step, all first-hop relations of every seed have to be inspected and categorised. In the worst case, if all seeds have disjoint first-hop relations and are fully connected, this takes $n_s|E|$ steps, where n_s is the number of seeds. During each step, one of three classes has to be selected. As an insight into real-world first-hop relation counts, Table 4 lists the degree $\deg(s)$ for each app's seed. We find that, as discussed in the seed identification step before, that seeds are typically highly connected with degrees roughly around half of $|E|$.

Decisive Role Assignment Regarding interaction cast, the domain expert selects decisive roles for every attribute of every entity in a seed partition. For each attribute, all roles that lie on a path to that entity need to be considered. In the worst case of a fully connected graph, this requires $O(|E|)$ steps and $O(|E|)$ decisions each. Computationally, the selection of decisive role candidates requires either a path enumeration or reachability checks, which, as discussed at the end of Sect. 3.4, result in factorial or quadratic cost, respectively.

The selection of decisive roles can be partly automated if an entity has only a single role candidate. Integral-only cases can be auto-assigned to a generic user role, as can be conditional-only cases with additional info about the condition. Otherwise the domain expert has to manually select from the list of candidates. Table 5 provides the number of single-candidate and integral-only entities as well as the average number of candidates available for the non-automatable entities.

Table 4. Evaluation of automatic steps for sample data models.

Application	**Full Scheme**			**Partition P_s**				**Ident. Cls.**			**Paths**	
	$\lvert E\rvert$	$\lvert R\rvert$	D[%]	$\lvert P_s\rvert$	$\lvert R_s\rvert$	D[%]	$\deg(s)$	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{D}\cap\mathcal{S}\rvert$	Ded.	Shared
Bugzilla 5.0	76	102	1.79	61	100	2.73	35	46	47	32	175	105908
Gitlab 12.7.5 CE	308	671	0.71	289	669	0.80	128	269	282	262	2997	>1M
Mattermost 5.18	40	54	3.46	32	54	5.44	27	28	23	19	41	1439
Taiga 5.5.7	68	116	2.55	61	114	3.11	29	54	56	49	238	103943
Zulip 3.2	77	116	1.98	66	116	2.70	46	45	54	33	54	226690

Table 5. Decisive role selection cost indicators for two practical examples.

Application	$\lvert P_s\rvert$	$\lvert R_s\rvert$	$\deg(s)$	Roles	Integral	Single Cand.	Avg. Cand.
Chat App Example	4	5	4	3	1	0	3
Mattermost	32	54	27	13	10	5	9

PD Listing If grouping terms cannot be automatically derived from already available display names or descriptions, an expert has to manually assign $|E|$ terms if entities are not aggregated. Besides the grouping term assignment, the list construction is automatic. The construction inspects every entity and adds their attributes or grouping terms to the four sets of every role. With a worst case of $|E|$ roles this leads to a $O(|E|^2)$ complexity.

4.2 Degree of Condensation

The degree of condensation depends of cause on the composition of the data model. Our approach condenses attributes for which a role is not decisive. Consequently, the degree of condensation is inversely proportionate to the ratio of decisive roles to role candidates. Hence, if every role is decisive for every connected attribute, the condensation would be minimal. Applying this worst case to the chat example from Table 2, the attributes of Channel and Post would be listed for each of the three roles resulting into 14 attributes and zero grouping

terms instead of the original 6 and 5. The condensation by the grouping terms depends on their defined scope, in this example an entity. Larger grouping scopes naturally result into fewer grouping terms per subject category.

5 Integration into Development Workflows

We argue that the best way to keep PD compliance documentation up-to-date and in sync with the app as its source of truth is to integrate Schemalyser into the development workflow. Thereby, vendors could offer PD listings like in Table 2 in a machine readable form as templates for customers' compliance documentation. In the following, we describe, how, by using code annotations, necessary domain knowledge could be added to further increase automation, and how Schemalyser can be used to monitoring changes to the data model.

Annotation Our approach relies on expert knowledge about the app architecture, mainly to determine and select decisive roles. Ideally, this information is noted by experts in a machine-readable way, e. g., in the form of code annotations, such that tools like ours can utilise it. Listing 1.1 shows how such an annotation might look like for the Channel class of the running example in Python. The `roledef` annotation defines a (FK) attribute as a new role. The `role` annotation assigns such a defined role as decisive role for the given attribute.

Listing 1.1. Exemplary code annotion for roles and assigned decisive roles.

```
class Channel:
    creator_id: int    # roledef: ChannelCreator, default
    created_at: int
    description: str   # role: ChannelMember
    lastpost_at: int   # role: Poster
```

Compliance Monitoring As a way to review or monitor compliance during development, Schemalyser could be integrated into testing: While tests are setting up a DB, a scheme could be dumped, pushed to the Schemalyser service, where it is compared to dumps of previous test runs and changes are flagged for review to ensure that additions of PD or identifying relations are intentional and compliant. Such change detection reapplies previous classifications to the new scheme whereupon new first-hop relations or role candidates trigger a review.

6 Conclusion

We have proposed a novel semi-automatic process to compile PD and subject categories, and condensed PD listings on the basis of app data models and entity relations. By analysing real-world app data models, we have pointed out the need for this condensed listing of PD to counter the effects of ubiquitous

identifiability in data models, where over 80 % of entities are attributable to data subjects. We argue that correctly assigning PD to subject categories requires a degree of architectural knowledge that is likely exclusive to the software vendor. We encourage vendors to add annotations about subject roles to their source code and follow our decisive role approach to make their knowledge accessible to customers in a machine readable way and allow a further automation of compiling comprehensive and up-to-date compliance documentation.

Acknowledgements The work is supported by the German Federal Ministry of Education and Research (BMBF) as part of the project Employee Privacy in Development and Operations (EMPRI-DEVOPS) under grant 16KIS0922K.

References

1. Andersson, M.: Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering. In: Entity-Relationship Approach — ER '94 Business Modelling and Re-Engineering. Ed. by P. Loucopoulos. Red. by G. Goos, J. Hartmanis and J. Leeuwen, pp. 403–419. Springer Berlin Heidelberg, Berlin, Heidelberg (1994)
2. Bercic, B., and George, C.: Identifying Personal Data Using Relational Database Design Principles. International Journal of Law and Information Technology 17(3), 233–251 (2009)
3. Burkert, C., and Federrath, H.: Towards Minimising Timestamp Usage In Application Software - A Case Study of the Mattermost Application. In: Pérez-Solà, C. (ed.) Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2019 International Workshops, DPM 2019 and CBT 2019, Luxembourg, September 26-27, 2019, Proceedings. LNCS, vol. 11737, pp. 138–155. Springer, Heidelberg (2019)
4. Fakas, G.J., Cawley, B., and Cai, Z.: Automated Generation of Personal Data Reports from Relational Databases. J. Info. Know. Mgmt. 10(02), 193–208 (2011)
5. Greiner, S., Buchmann, T., and Westfechtel, B.: Bidirectional Transformations with QVT-R: A Case Study in Round-Trip Engineering UML Class Models and Java Source Code. In: Hammoudi, S. (ed.) MODELSWARD 2016 - Proceedings of the 4rd International Conference on Model-Driven Engineering and Software Development, Rome, Italy, 19-21 February, 2016, pp. 15–27. SciTePress (2016)
6. Huth, D., Tanakol, A., and Matthes, F.: Using Enterprise Architecture Models for Creating the Record of Processing Activities (Art. 30 GDPR). In: 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC), pp. 98–104. IEEE, Paris, France (2019)
7. Martin, Y.-S., and Kung, A.: Methods and Tools for GDPR Compliance Through Privacy and Data Protection Engineering. In: 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), pp. 108–111. IEEE, London, United Kingdom (2018)
8. Ryan, P., Pandit, H.J., and Brennan, R.: A Common Semantic Model of the GDPR Register of Processing Activities. In: Serena, V., Harasta, J., and Kremen, P. (eds.) Legal Knowledge and Information Systems - JURIX 2020: The Thirty-Third Annual Conference, Brno, Czech Republic, December 9-11, 2020. Frontiers in Artificial Intelligence and Applications, pp. 251–254. IOS Press (2020)