Consent verification under evolving privacy policies

Marco Robol[1], Travis D. Breaux[2], Elda Paja[3], Paolo Giorgini[1]
Information Engineering and Computer Science Department, University of Trento, Trento, Italy[1]
Institute of Software Research, Carnegie Mellon University, Pittsburgh, USA[2]
Computer Science Department, IT University of Copenhagen, Copenhagen, Denmark[3]
{marco.robol,paolo.giorgini}@unitn.it, breaux@cs.cmu.edu, elpa@itu.dk

***Abstract* — Personal data provides important business value, for example, in the personalization of services. In addition, companies are moving toward new business models, in which products and services are offered without charge to users, but in exchange for targeted advertising revenue. New privacy regulations require organizations to explicitly state their data practices in privacy policies, including which data types will be collected. By consenting to data collections described in a policy, the user acknowledges that he or she is granting the company the authorizations needed to access their data. When data practices change, a new version of the policy is released. This release can occur a few times a year, when requirements are rapidly changing for the collection and processing of personal data. Furthermore, the user may change his or her privacy consent by opting in or out of the policy. We propose a formal framework to support companies and users in their understanding of policies evolution under consent regime that supports both retroactive and non-retroactive consent and consent revocation. Preliminary results include an ontology for policy evolution, expressed in Description Logic, that can be used to formalize consent and data collection logs and then query for which data types can be legally accessed.**

***Index Terms* — Privacy, consent, policies, evolution, formal framework, description logic.**

I. Introduction

Consent is a key element in privacy, and it has become a critical element under the EU General Data Protection Regulation (GDPR). Under GDPR, consent is one of a few legal bases available that can be used to process user data (see Article 6-1(a)) and, in most cases, consent is the only viable basis. Consent constitutes legal evidence of user awareness about their data being collected, used, and shared by companies. Under the GDPR, the user is protected because the demonstration of a valid acquisition of the consent is a responsibility of the company, i.e., the user does not need to initiate a request to receive this protection. To this end, the company must present information about how the data will be processed, and then request consent from the user before processing the data. Furthermore, the user may revoke their consent at a later date, which means the company can no longer process data collected after the revocation (see Article 7). However, the company may continue to process data collected under the previously granted consent, if they choose to do so. In addition to privacy policies, consent can be obtained in other ways prior to collection, including just-in-time consent after a user has already begun using the service, but prior to taking an action within the service [14].

User decisions about granting consent can be driven by the perception of trust in a company with respect to their history of bad privacy practices. For example, recent disclosures by Facebook of leaks of personal data to third-parties [9] can lead some users to restrict their privacy preferences, which are controls on the accessibility of their data, or to delete their account, thus opting out entirely.

Internally, organizations may make changes to their data practices several times a year. Evidence of changes can be observed in the revision histories of evolving privacy policies. Figure 1 shows the changes to the privacy policy of Waze, a popular mobile app for automobile navigation: the y-axis shows the number of statements per policy revision, with the policy revision dates along the x-axis; exact statement matches appear in blue, new statements appear in red, and statements with changes to wording appear in orange. Some of these changes are due to changes in boilerplate language (e.g., how the company or user are referenced), or to data purposes. Under the GDPR, changes to data practices and purposes require consent. While Waze in particular underwent a number of changes from late 2012 to mid-2014, there were significant changes from late 2017 through mid-2018, at which point the GDPR went into effect.

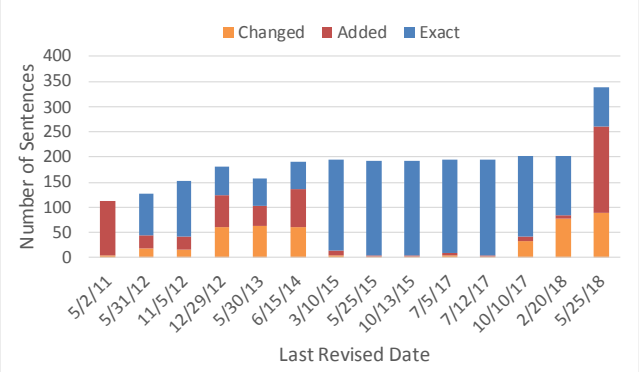

Figure 1. Revision History of the Waze Privacy Policy from 2011-2018

As a matter of requirements engineering, under the GDRP companies must tag their data to know when it was collected and when they obtained consent. Because the GDRP requires that consent be granular, including that purposes be distinguishable (see Recitals 32 and 43), companies should also tag this data with purposes for which consent was granted. Notably, companies may collect data as a consequence of their system design but they may not process the data for a specific purpose without consent. At scale, one can imagine that companies who are in competitive markets will be looking for new opportunities to process user data, leading to changes to their practices. In addition, users may either be uncomfortable with new purposes, or shift their trust in companies due to improper data handling

2332-6441/19/$31.00 ©2019 IEEE
DOI 10.1109/RE.2019.00056

IEEE computer society

by the company or the market. To address this problem, we propose a formal framework, expressed in description logics (DLs), to support users and companies in the understanding of consent under evolving policies. The contributions of the paper include an ontology for privacy evolution that can be used by companies and users to reason about data access under multiple consent granting and revocation scenarios.

II. Challenges

Description logics (DLs) [4] are the de-facto languages for ontologies and the Semantic Web. DLs are a subset of first-order logic languages, less expressive, but that guarantee resolvability. We chose DL because of its ontology orientation, since we need to represent and verify hierarchical relationships between the data types used in privacy policies. For example, users can provide consent on some broad category of data, such as, *personal information*, or they can provide separate consents on narrower categories, such as, *e-mail address* or *phone number*. These hierarchical relationships can lead to inconsistencies and conflicts in deciding if data can be processed [6]. We propose the use of DLs to build consistent taxonomies of data types used in privacy policies. However, in the formalization of consent evolution, DLs have limited expressivity and an open world assumption, which introduce specific challenges.

Description logics have a limited expressivity, that has been introduced in favor of decidability [12]. For example, relations are binary, only, and cannot be extended to include other dimensions, such as temporality. This poses a challenge when extending the static policy representations to address evolution.

The open world assumption means that unknown facts are considered neither false nor true. In contrast, a closed world assumption, also called negation by failure, considers unknown facts as false by default. In DL, if one assumes that "consent given" is equivalent to "consent not withdrawn," without explicitly declaring these concepts as equivalent, then one would draw incorrect conclusions from the knowledge base. Note that, this means that a policy can either be consented to or not, but it can also be in an unspecified consent state. It is important to keep this in mind also when it is time to query the knowledge base. For example, non-consented policies can be queried by looking for policies that are not consented, however, if consent is unspecified, the policy will be missed in the results of the query.

Monotonicity is a desirable property for update functions of any formalization, because additions to a knowledge base with this property do not invalidate prior facts. Monotonic update functions are simpler and more efficient, and ensure that the existing knowledge base is never changed but only extended with new facts. However, consent evolution appears to have a non-monotonic behavior. For example, in the case of a withdraw event, the update function should change the consent status from approved to withdrawn, in a non-monotonic way. To ensure monotonicity, we choose a different representation of evolution. For consent, we use intervals of time that define the validity of consent over time. Consent is not withdrawn if the concept is still valid at a specific time, otherwise, it is withdrawn if it was valid at some time in the past, but not anymore.

Representation of time is not supported in DLs. There are temporal extensions for DLs [3] that increase expressive and computational complexity, which we want to avoid in favor of understandability and efficiency. For example, some temporal extensions are based on the representation of many time-related concepts, relationships, and constraints, such as a specific time instant, time interval, the concepts of before, after, meanwhile, started before, ended after, etc.. Moreover, the representation of temporal concepts, such as the interval of time in which consent was granted and then withdrawn, can be approached in different ways. A simplification to avoid these challenges is to focus only on forward-time or backward-time. However, restricting the time representation in consent evolution to either forward- or backward-time is too limited, because we aim to support retroactive consent approval and retroactive consent withdrawal, which require backward- and forward-time, respectively.

III. Formalization

This section presents the formal framework on consent evolution. The formal framework is specified in description logic (DL) and includes key concepts to express policies, data types, data collection, consent, users, and time. In the notation that we use in this paper, lowercase terms are used for individuals and capital letters are used for concepts, where individuals belonging to concepts is expressed with the symbol $\in$, for example $d \in D$, then we use the symbol $\sqsubseteq$ to express subsumption between concepts, and the symbol $\sqcap$ to express intersection of concepts. The following sections present the main concepts of this framework.

A. Policies

A privacy policy is a set of desired authorizations, needed by organizations to access and use data about users. Such authorizations are granted by each user for his own data. Organizations create these policies to cover the purposes for which they use data in their business. While policies are static and cannot be modified, new versions can be created. In the US, privacy policies typically include statements that users will agree to new versions of the policy. Under GDPR, users have the right to opt-in and thus companies can no longer assume that users are covered by the new policy. Changes to a policy occur for several reasons, including: (1) to make terminology consistent with legal practices; or (2) to describe a new or modified service. In our formalization, we focus specifically on changes of kind (2) that affect authorizations to access and use data. Each authorization specifies a data type and a modality, which we now discuss.

1) Data types

Data types, such as, e-mail address, are classes of data used in policies to define the scope of an authorization. In the taxonomy of data used in a policy, data types can subsume other data types, for example, contact information subsumes email address and phone number. Parent types in the taxonomy represent classes of data that are broader than children types. In a policy, broader types are used to allow more flexibility, while narrower types of data are used to decrease policy vagueness.

In description logic, data types are concepts that are subsumed by the concept $Data$. For an arbitrary data type D,

this is written as $D \sqsubseteq Data$. Instances of data, such as an e-mail address or a record in a database, are individuals members of some data type, written $d \in D$, for an arbitrary data instance d.

2) Modalities

Modalities are the data actions that can be authorized. In privacy, it is common to refer to three main modalities: collection, use, and sharing. In this paper, we focus on use, thus, all policies, and the authorizations considered herein, are implicitly about use. While not collecting any data can eliminate privacy risk from the beginning, several mitigations can be adopted to reduce risk in a later phase by limiting the use of data, in the case where collection cannot be prevented because of software design. Moreover, the third modality, sharing, deserves special attention, because it can introduce re-purposing [7]. On this argument, we plan to go into more details in future work.

In our formalization, we represent authorizations with a single relationship, $authorizes$ from a policy to a data type. Policies are individuals of the class $Policy$ that authorize data, for example, $p \in Policy, p\ authorizes\ D$.

B. Data collection

User data is collected by organizations and then used to provide or improve the quality of services. Data can be collected from users or third parties. Consent-based authorizations must account for time of collection, because consent can *minimally* be granted for data collected after the time of consent. On the other hand, consent can *maximally* be granted for data collected after and also before the time of consent, which is called retroactively as we will discuss later. Thus, depending on the collection time and type of consent, some data may not be accessible.

The *collection log* is used to maintain a record of compliance with user consent. The log is continuously updated each time a data element is collected. Each log entry consists of the data type, collection time, and the user from whom the data was collected. In description logic, the collection log is represented by individuals of sub-classes of $Data$, for example, $d \in D$ for $D \subseteq Data$. Collection time is expressed using the relationship $collectedAtTime$ and the user of the data is expressed by the relationship $ofUser$. For example, $user@gmail.com \in Email \subseteq Data$, $collectedAtTime\ t1$, and $ofUser\ u1$.

C. Consent

Consent approval is the acknowledgement by the user to grant an authorization desired by the company as specified in the privacy policy. Specifically, by consenting to a policy, the user grants the authorizations on his or her data. Consent that only authorizes use of data collected in the future is called non-retroactive consent. Retroactive consent grants authorizations in the policy for both newly and previously collected data. Retroactivity is important, because it could be the only way for organizations to access historical data and to re-purpose this data under a new policy. However, it could also be dangerous, because data hidden by a user from his or her past could suddenly become accessible under a retroactive consent.

Consent withdrawal is the opposite of consent approval. When consent is withdrawn, previously granted authorizations in a policy are revoked. Similar to approval, a non-retroactive withdrawal means data collected under a previous authorization can still be processed, however, the authorizations are no longer valid for newly collected data. A non-retroactive withdrawal can be dangerous for the user, because data authorized in the past will remain accessible in the future. Retroactive withdrawal means authorizations are no longer granted for data previously or newly collected. Withdrawing consent only revokes authorizations granted by the previous approval. Authorizations granted through other consents remain untouched.

It worth mentioning that the *right to be forgotten* proposed in the GDPR is similar to retroactive withdrawal, but with some differences. In the case of retroactive withdrawal, access is lost to the data, but it may be obtained again though the acquisition of a new consent, whereas in the right to be forgotten requires that data are deleted and cannot be accessed later, even in the event of a new retroactive consent by the user.

1) Evolving consent

Consent can be given and withdrawn many times, but each pair of events of approval and withdrawal defines an independent interval of time in which consent is given. The *consent log* includes all the events of consent approval and withdrawal for every user. The consent time intervals are observable in the consent log. Each interval is represented by: (i) the times of approval and withdrawal, (ii) the user who provides the consent, (iii) the policy consented to, (iv) the retroactivity of approval and withdrawal. Consent authorizes data access by collection time, depending on retroactivity and consent time and withdrawal time.

Figure 2 shows the interactions of four states: non-retroactive and retroactive consent approval and withdrawal. For each state, the horizontal shaded bars show the authorization by collection time: the dark shading shows where data collected at a specific time is accessible; the light shading shows were data is inaccessible. The vertical lines show times where consent is approved or withdrawn. Under GDPR, companies are permitted to use non-retroactive consent and non-retroactive withdrawal.

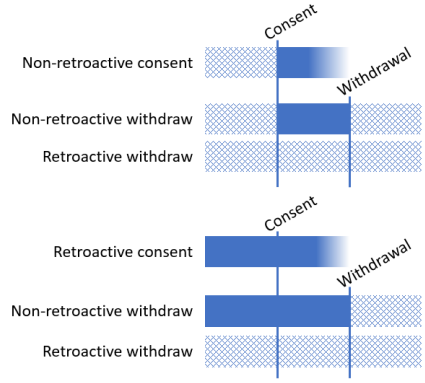

Figure 2. Four States of Retroactivity

In our DL formalization, consented intervals are individuals of the class $Consent$, where the policy on which consent is given is specified with the relationship $onPolicy$, the user that has consented is specified with the relationship $byUser$, and the times in the interval are specified with the relationship $atTime$. Consent times are all the times included in the interval between approval and withdrawal. Different classes of consent intervals exist, based on retroactivity as follows:

- $NonRetroactivelyGiven \subseteq Consent$
- $RetroactivelyGiven \subseteq Consent$
- $NonRetroactivelyWithdrawn \subseteq Consent$
- $RetroactivelyWithdrawn \subseteq Consent$

The above classes are consented intervals, in which $NonRetroactivelyGiven$ means that the interval starts with a non-retroactive consent, while $RetroactivelyWithdrawn$ means that the interval ends with a retroactive withdrawal. Consent cannot be both retroactive and non-retroactive, and the same is true for withdraw. Thus, the following axioms are true:

- $NonRetroactivelyGiven \sqcap RetroactivelyGiven \equiv \emptyset$
- $NonRetroactivelyWithrawn \sqcap RetroactivelyWithdrawn \equiv \emptyset$

D. Time

Time is a critical concept in formalizing evolution. When consent evolves, it may apply to new or old data. New data is when collection time came after consent time, and old data is when collection time came before consent time, where *before* is the inverse of the temporal concept *after*. In our DL formalization, times are individuals of the concept $Time$. The relationships of $afterThen$ and $beforeThen$ are used to express relative time-order. These relationships are both reflexive, transitive, and inverse to one another as follows: $afterThen\ inverseOf\ beforeThen$. For example, we can have two times $t1$ and $t2$, so that $t1\ beforeThen\ t2$; and we can refer to all the times before $t2$ with the following equivalence class: $Time\ and\ beforeThen\ t2$.

E. Updating the knowledge base

In DL, a knowledge base (KB) consists of axioms expressed over concepts, relations and individuals. In our work, the KB is comprised of the collection log and the consent log. Since these grow over time, we must easily update the KB. To do so, we provide update functions for each of the main events. The update functions are intended to be called following the order in which events occur, whereas applying the functions to non-temporally ordered events can result in an inconsistent KB. The main events for which the KB is updated are: (i) creation of a policy, (ii) data collection, (iii) consent approval, (iv) consent withdrawal. A description of the main update functions in pseudo-code follows:

Listing 1: Creation of a new policy

1	*Declare a new individual $p \in Policy$;*
2	*For each authorization, assert $authorizedByPolicy$ relationship from the authorized data type to the policy.*

Listing 2: Creation of a new time

1	Declare a new individual $t \in Time$;
2	Assert $afterThen$ with the previous time;
3	Assert $includesTime$ with all consents not yet withdrawn.

Listing 3: Data collection

1	*Given the data type D of a collected data, declare a new individual $d \in D$;*
2	*Assert $ofUser$ relationship with user of the data;*
3	*Assert $collectedAtTime$ relationship with current time.*

Listing 4: Consent approval

1	Depending on retroactivity, declare an individual either: $c \in NonRetroactivelyGiven$ or $c \in RetroactivelyGiven$;
2	Assert $onPolicy$ relationship to consented policy;
3	Assert $includesTime$ relationship with current time;
4	Assert $byUser$ relationship with consenting user.

Listing 5: Consent withdrawal

1	If retroactive: $c \in RetroactivelyWithdrawn$;
2	Else: $c \in RetroactivelyWithdrawn$.

F. Querying

The querying system is based on DL expressions, which return equivalent classes, sub-classes and individuals in the interpretation of a query expression. In our work, the individuals in the KB are those contained in the collection and consent logs.

We now present how to query the KB for authorized data. We provide four subqueries, one for each configuration of consent retroactivity. The queries return data subsets by collection time, e.g., the subquery in which a non-retroactive consent is non-retroactively withdrawn, returns data that has been collected within the intervals of time delimited by a consent and its withdrawal, intervals that we call consented intervals.

The final query, given a current time *t* (now) and a consent *c* given by a user *u* on a policy *p*, returns the set of data that are: (i) authorized in the policy *p*, (ii) by the user *u*, (iii) controlled by some preference that is opted-in (at time *t*) as part of consent *c*, and (iv):

1. (if c is still consented) collected *after the approval*;
2. (if consent c has been retroactively given and then non-retroactively withdrawn) collected *before the withdrawal*;
3. (if consent *c* has already been non-retroactively withdrawn) collected *after the approval but before the withdrawal*;
4. (if consent *c* has been retroactively given and it is still consented) independently from collection time.

Figure 3 shows the classes of data (by collection time) that are returned by each of the four sub-queries.

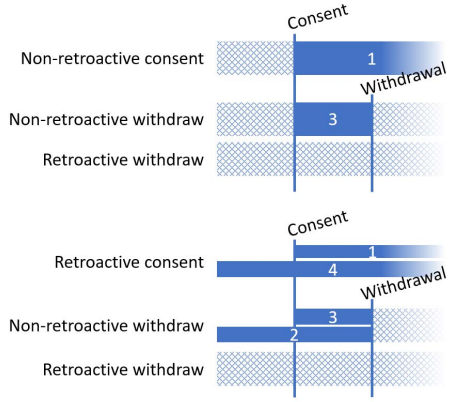

Figure 3. Data Classes Returned by Sub-Queries

Listing 6 is the DLs expression for the sub-query 2, of data from user *u1* authorized by consent on policy *p1*, authorized at time *t7*.

Listing 6: Sub-query n.2: user *u1*, policy *p1*, at time *t7*.

```
Data and (authorizedByPolicy value p1) and (ofUser value u1) and
(collectedAtTime some (Time and beforeThen some
  (Time and ( inverse (includesTime) some
    (Consent and RetroactivelyGiven and Withdrawed
     and (givenByUser value u1) and (onPolicy value p1)
    )
  ) )
) ) and
(inverse (activatesData) some
  (Preference and activeAtTime value t7 and actsOnConsent some
    (Consent and RetroactivelyGiven and Withdrawed
    and (givenByUser value u1) and (onPolicy value p1)
    ) ) )
```

In the KB, the collection and consent logs are decoupled, so that retroactivity of consenting and withdrawing are evaluated, toward collected data, at query-time, which allows for a monotonic update of consent and data in the KB.

IV. Worked example

This section presents four scenarios of policy and consent evolution, using the formalism to support the reader in a better understanding of the scenario itself.

A. Organization expansion

A bus company is going to introduce a new service, which will allow the users to use their smartphone to check the routes and the position of the busses in real time using a mobile app. This application will help the users to use the company busses and, meanwhile, it will allow the company to collect data to optimize the bus routes.

The bus company creates a new privacy policy that describe the purposes for using the new user data. The company acquires the new consent of the users directly through the app. Users not interested in using the app will remain under the old policy. However, the company allows users the possibility to deactivate access to their position data for statistical analyses with an opt-able-out preference.

$$Location \subseteq UserData \subseteq Data$$
$$pl \in Policy$$
$$Location\ authorizedBy\ pl$$
$$Preference\ and\ actsOnConsent\ onPolicy\ pl$$
$$and\ activatesData\ Location \subseteq activeAtTime\ consent\_time$$

B. Historical data

Users of the bus company have used rechargeable smartcards for years. The company has been recording all data from the smartcard system. Now, they want to use this data to analyze and optimize bus routes. To do so, a retroactive consent is acquired for the new policy to permit access to historical location data of the users and to re-purpose this data for route optimization.

$$Consent\ onPolicy\ pl \subseteq RetroactivelyGiven$$

C. Opt-able-in privacy preference

Due to the increased user discomfort from being tracked by the app, the bus company changes the policy again. In the new policy, the use of location data for routing optimization is by default turned off, even if the data is still being collected. The user can manually opt-in to the preference to authorize the company to use their location data, which includes the data from the app and additional data provided by third parties.

$$ThirdPartyLocation \subseteq ThirdPartyData \subseteq Data$$
$$ThirdPartyLocation\ authorizedBy\ pl$$
$$Preference\ and\ actsOnConsent\ onPolicy\ pl$$
$$and\ activatesData\ ThirdPartyLocation$$
$$\subseteq not\ activeAtTime\ consent\_time$$

D. Consent withdrawal

After a major scandal for data breaches that involve the bus company, concerned users begin to retroactively withdraw their consent to the bus company policy after every use of the mobile app. In order to use the app, users are still asked to consent the policy, however. By using a non-retroactive consent, they limit the access to only new data. Doing so, the app user experience lacks all the data-driven functionalities, such as preferred routes, notifications of preferred buses delays, and bus suggestions based on user schedule.

$$ConcernedUserConsent \subseteq NonRetroactivelyGiven$$
$$ConcernedUserConsent \subseteq RetroactivelyWithdrawn$$

V. Related Work

Description logics (DLs) are a subset of first order logic, intended as a general purpose language for knowledge representation, where decidability is valued over expressiveness. The components of description logic are: (i) concepts, (ii) their relations or properties, and (iii) individuals. When using DL to represent an application domain, definitions of concepts and properties compose the TBox, while assertions about individuals and their concepts and properties compose the ABox.

A. Temporality in description logic

Temporal representation is not directly supported by DL. However, time and temporal concepts can be modeled. The OWL-Time ontology [10] provides concepts related to time representation, however it does not specify how to use these concepts, nor how to reason over such concepts. In general, temporal representation focuses on instants and/or intervals. In a point-based representation, relations between instants are "before", "after", and "equals." In an interval-based representation, relations can get up to the 13 pairwise disjoint Allen's relations [1] showed in Figure 4.

In the case of numerical representation of time, Allen's relations can be easily inferred. For qualitative representation, by assertion of Allen's relations, inferring non-declared relations or checking consistency is an NP-hard problem.

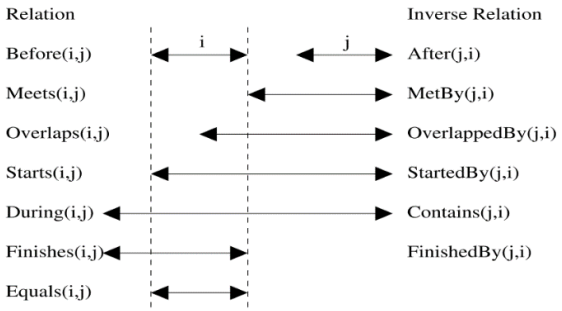

Figure 4. Allen's temporal interval relations

Apart from missing representation of time, DL formalisms also miss constructs to represent the evolution of concepts and their properties in time. Because DL only supports binary relations, temporality is not easily encoded as a dimension of an existing relation. For example, a data collection can be expressed as a relation between a data type and a user, however to include also the time of collection we would need a ternary relation.

Many approaches have been proposed to address such problems [3]. Versioning has been discussed in [11], which proposes to create a copy of the ontology at every change. The n-ary relations approach [13] and 4D-fluents [16] are two alternative approaches to represent evolution of concepts. N-ary suggests representing a temporal ternary relation (object, verb, predicate, time) as a concept itself representing the verb, with properties to relate it to the object, the predicate, and the time. The 4D-fluents approach represents temporal relation as a 4-dimensional object, which includes time-specific temporal versions of the object and the predicate, where the original relation is now expressed between the temporal versions of the concepts. With respect to the n-ary approach, 4D-fluents suffers from proliferation of objects (two additional objects for each temporal relation). While n-ary suffers from data redundancy in the case of inverse and symmetric properties (e.g., the inverse of a relation is added explicitly twice).

In the representation of consent evolution, collection defines a time instant and consent defines an interval between consent approval and withdrawal. Our approach is based on the representation of non-overlapping time intervals, where data are collected within one of these intervals, while a sequence of intervals defines a consent.

B. Privacy preferences and user personalization

Recently, companies have pursued advanced personalization of user experiences to strengthen their relationship with users [2,5,15]. Personalization is commonly intended as customized or customizable user experiences, based on user's behavior or preferences. Privacy preferences are explicit requests by users about how or when their personal data will be used. For example, users may want to exclude their browser search history from the dataset used by websites to show targeted advertisements.

The P3P [8] is the reference platform for privacy preferences on the web. With P3P, privacy agents are in charge of evaluating website privacy policies with respect to user's privacy preferences. However, preferences are never used to modify websites privacy settings.

VI. Conclusion

We presented a framework for the representation of consent under evolving policies to support companies in better understanding how policy changes affect their ability to access data in a compliant manner, within a consent-based context. This work represents an attempt to show the complexity of policy evolution and demonstrate the ability of description logic in modeling this domain. Future work includes an evaluation using a real case study that considers how user consent- and preference-choices could be affected by their perception of privacy mitigations. Where users may be more concerned about not using specific data types for specific purposes, companies may want to separate those data types into separate policies to avoid having users opting-out entirely. An additional direction of research consists of extending the formalization to include an access log, which allows one to automatically verify the compliance of accesses. We also plan to study repurposing, which requires one to include the modality of sharing in the formalism. Finally, future work must address the granularity of consent, a fundamental requirement of the GDPR, that can be supported by privacy preferences.

REFERENCES

[1] J.F. Allen. "Maintaining knowledge about temporal intervals." *Comm, of the ACM*, 26.11(1983):832–843.

[2] A. Ansari, S. Essegaier, R. Kohli, "Internet recommender systems," *J. Marketing Research*, 37 (2000): 363-375.

[3] A. Artale, E. Franconi. "A survey of temporal extensions of description logics." *Ann. of Math and AI* 30.1-4(2000): 171-210.

[4] F. Baader, I. Horrocks, U. Sattler. "Description logics." *Handbook on ontologies*. Springer, 2004. 3-28.

[5] M. Berry, G. Linoff, *Data mining techniques for marketing, sales and customer support*, Wiley, NY, 1997.

[6] T. D. Breaux, H. Hibshi, A. Rao. "Eddy, a formal language for specifying and analyzing data flow specifications for conflicting privacy requirements," *Req'ts Engr. J.*, 19(3): 281-307, 2014.

[7] T. D. Breaux, D. Smullen, H. Hibshi. "Detecting repurposing and over-collection in multi-party privacy requirements specifications." *23rd Int'l Req'ts Engr. Conf.*, pp. 166-175, 2015.

[8] Cranor, Lorrie. *Web privacy with P3P*. O'Reilly, 2002.

[9] C. Cadwalladr, E. Graham-Harrison. "The Cambridge analytica files." *The Guardian* 21 (2018): 6-7.

[10] J.R. Hobbs, F. Pan. "Time ontology in OWL." W3C working draft 27 (2006): 133 (https://www.w3.org/TR/owl-time).

[11] M. Klein, D. Fensel. "Ontology versioning on the semantic web." *Int'l Semantic Web Working Symp*. pp. 75–91, 2001.

[12] H.J. Levesque, R.J. Brachman. "Expressiveness and tractability in knowledge representation and reasoning 1." *Computational intelligence* 3.1 (1987): 78-93.

[13] N. Now, A. Rector, P. Hayes, C. Welty. "Defining N-ary Relations on the Semantic Web." *W3C W' Group Note*, 12, 2006

[14] F. Schaub, R. Balebako, A. L. Durity, L. F. Cranor, "A design space for effective privacy notices," *Symp. on Usable Privacy and Security*, 2015.

[15] M. Spiliopoulou, "Web usage mining for web site evaluation – making a site better fit its users," *Comm. of the ACM*, 8(43): 127-134, 2000.

[16] C. Welty, R. Fikes. "A reusable ontology for fluents in OWL." *4th Int'l Conf. Formal Ontology in Inf. Sys.*, pp. 226– 336, 2006.