# Detecting Repurposing and Over-Collection in Multi-party Privacy Requirements Specifications

Travis D. Breaux[1], Daniel Smullen[1], Hanan Hibshi[1, 2]
Institute of Software Research, Carnegie Mellon University[1]
Pittsburgh, Pennsylvania, United States
College of Computing, King Abdul-Aziz University[2]
Jeddah, Saudi Arabia
{breaux,dsmullen,hhibshi}@cs.cmu.edu

***Abstract*—Mobile and web applications increasingly leverage service-oriented architectures in which developers integrate third-party services into end user applications. This includes identity management, mapping and navigation, cloud storage, and advertising services, among others. While service reuse reduces development time, it introduces new privacy and security risks due to data repurposing and over-collection as data is shared among multiple parties who lack transparency into third-party data practices. To address this challenge, we propose new techniques based on Description Logic (DL) for modeling multi-party data flow requirements and verifying the purpose specification and collection and use limitation principles, which are prominent privacy properties found in international standards and guidelines. We evaluate our techniques in an empirical case study that examines the data practices of the Waze mobile application and three of their service providers: Facebook Login, Amazon Web Services (a cloud storage provider), and Flurry.com (a popular mobile analytics and advertising platform). The study results include detected conflicts and violations of the principles as well as two patterns for balancing privacy and data use flexibility in requirements specifications. Analysis of automation reasoning over the DL models show that reasoning over complex compositions of multi-party systems is feasible within exponential asymptotic timeframes proportional to the policy size, the number of expressed data, and orthogonal to the number of conflicts found.**

***Index Terms*—Data flow analysis, privacy principles, requirements validation.**

## I. Introduction

Increasingly, companies expose their information system features to outside developers for reuse and integration into third-party applications through Application Programmer Interfaces (APIs). Popular examples include the Google Maps API for navigation and routing services, and the Facebook Login and Google+ for identity management and social networking services. These services reduce development costs because outside developers can leverage stable third-party code. When these services are offered at low or no cost, small companies and end-user programmers can trial these services while developing cutting-edge application concepts, such as the Waze crowdsourcing traffic application that combines social networking with first-person traffic monitoring (e.g., reporting road hazards, speed traps, etc.)

Increased integration has obvious benefits to developers and consumers alike; however, new privacy risks arise from increased information sharing across services. Information that users share with friends on Facebook, for example, can be exposed to anonymous Waze users, or third-party advertisers with whom the Facebook users have no direct business relationship. This includes receiving access to a user's social network data or to a user's travel itinerary. Despite increased E.U. and U.S. government regulations [7], to our knowledge software developers lack the tools needed to specify personal data requirements in a distributed system that align their practices with accepted privacy principles.

**Contributions**: we present new techniques to formally model multi-party data flows requirements and align these models with three critical privacy principles: the *purpose specification principle*, which requires that the purposes for which data is collected should be explicitly stated; the *collection limitation principle*, which requires that collection of personal data should be limited, e.g., limited to that which will be used; and the *use limitation principle*, which requires that uses be limited to the purposes for which the data was originally collected (with exceptions for consent and legal compliance). These principles are represented in varying degrees across the U.S. Fair Information Practice Principles (FIPPs) and OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data, and van der Sipe and Maalej note the impact of these principles on mobile applications, in particular [24]. Together, the principles reduce the risk of *over-collection*, which is the collection of more information than is needed, and *repurposing*, which is using or sharing data for purposes other than that for which data was collected. Repurposing occurs at data collectors, who collect information from the data subject, or at data processors, who process personal information on behalf of data collectors. In composable systems, these principles are challenging to verify due to the transitive nature of data storage and processing across multiple parties.

The remaining paper is organized as follows: we introduce a running example in Section II, the approach in Section III, our case study design in Section IV, with results in Section V, threats to validity in Section VI, related work in Section VII, and our discussion and summary in Section VIII.

978-1-4673-6905-3/15 © 2015 IEEE

RE 2015, Ottawa, ON, Canada
Research Paper

## II. Running Example

We illustrate our approach in a running example based on the mobile application (app) Waze, which is a navigation app that uses crowdsourcing to report traffic, road hazards, speed traps, and other events. We revisit this example throughout the paper and within our case study design and results.

Data requirements that govern the data flows seen in Fig. 1 appear in platform developer polices, customer agreements, terms-of-use and terms-of-service agreements, and privacy policies. Because many of these documents are viewed as legal contracts, developers must align these requirements with their information systems. In the case of privacy policies, the policy also serves as a representation to the user about what the company does internally with user data.

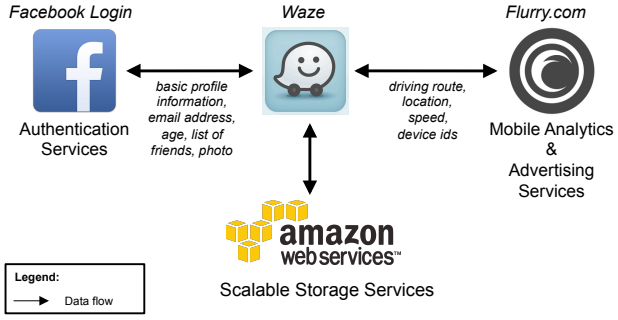


Fig. 1. Data flows among the Waze mobile app and their third-party service providers; social network data is collected from Facebook Login and potentially shared with Flurry.com, while Waze user data is stored in the Amazon AWS cloud

## III. Approach

We first review background, before introducing the new Eddy language extensions.

### A. Background on Privacy Specifications

Our privacy specifications are based on the Eddy language [8], which has formal semantics expressed in Description Logic (DL) - a subset of first-order logic for expressing knowledge. A DL knowledge base *KB* is comprised of intensional knowledge, which consists of concepts and roles (terminology) in the TBox $T$, and extensional knowledge, which consists of properties, objects and individuals (assertions) in the ABox [3]. In this paper, we use the DL family $\mathcal{ALC}$, which includes logical constructors for union, intersection, negation, and full existential qualifiers over roles. Concept satisfiability, concept subsumption and ABox consistency in $\mathcal{ALC}$ are PSPACE-complete [3].

Description Logic includes axioms for subsumption, disjointness, and equivalence with respect to a TBox. Subsumption describes individuals using generalities: we say a concept $C$ subsumes a concept $D$, written $T \vDash D \sqsubseteq C$, if $D^{\mathfrak{I}} \subseteq C^{\mathfrak{I}}$ for all interpretations $\mathfrak{I}$ that satisfy the TBox $T$. The concept $C$ is disjoint from a concept $D$, written $T \vDash D \sqcap C \rightarrow \perp$, if $D^{\mathfrak{I}} \cap C^{\mathfrak{I}} = \emptyset$ for all interpretations $\mathfrak{I}$ that satisfy the TBox $T$. Finally, the concept $C$ is equivalent to a concept D, written $T \vDash C \equiv D$, if $C^{\mathfrak{I}} = D^{\mathfrak{I}}$ for all interpretations $\mathfrak{I}$ that satisfy the TBox $T$.

The universe of discourse consists of the set *Req* of requirements, *Action* of actions, *Actor* of actors, *Datum* of data types, and *Purpose* of data purposes. A specification is a DL knowledgebase KB that consists of multiple requirements. A *requirement* is a DL equivalence axiom $r \in Req$ that is comprised of the DL intersection of an action concept $a \in Action$ and a role expression that consists of the DL intersection of roles $\exists R_1 \sqcap \ldots \exists R_n \in Roles$. We are primarily concerned with four roles in this paper: *hasSource* indicates the source actor from whom the data was collected; *hasObject* indicates the data on which an action is performed; *hasPurpose* indicates the purpose for which data is acted upon; and *hasTarget* indicates the recipient to whom data is transferred. For example, requirement $p_0$ for a $location \in Datum$, and purpose $providing\_services \in Purpose$ in the TBox $T$, such that it is true that:

$$(1)\quad T \vDash p_0 \equiv COLLECT \sqcap \exists hasObject.location \sqcap \exists hasSource.waze\_user \sqcap \exists hasPurpose.providing\_services$$

Each requirement is contained in exactly one modality concept in the TBox $T$ as follows: *Permission* contains all actions that an actor is permitted to perform; *Obligation* contains all actions that an actor is required to perform; and *Prohibition* contains all actions that an actor is prohibited from performing. We adapt the axioms of Deontic Logic [13], such that it is true that $T \vDash Obligation \sqsubseteq Permission$, wherein each required action is necessarily permitted. If the requirement $p_0$ is required such that $T \vDash p_0 \sqsubseteq Obligation$, then $T \vDash p_0 \sqsubseteq Permission$. We can now compare the interpretations of two requirements based on the role fillers to precisely infer conflicts. A *conflict* is defined as $Conflict \equiv Permission \sqcap Prohibition$.

We define a data flow *trace* as a subset of requirements pairs $(r_s, r_t) \in Req \times Req$ that map from a source action $r_s$ to a target action $r_t$. We can trace permitted data collections (source action) to permitted data uses and data transfers (target actions) when the role values of the source and target actor, datum and purpose entail a shared interpretation. For each requirement written in the form $r_i \equiv a \sqcap \exists R_{i,1}.F_{i,1} \sqcap \exists R_{i,2}.F_{i,2} \sqcap \ldots \sqcap \exists R_{i,n}.F_{i,n}$ in the TBox $T$, such that $a \in \{COLLECT, TRANSFER, USE\}$ and $R_{i,1} \ldots R_{i,n} \in Roles$, we compare role fillers $F_{i,1} \ldots F_{i,n}$ between the source and target actions to yield one of four exclusive *Modes* as follows for some $j \neq k$:

- *U: Underflow*, occurs when the data source is subsumed by the target, if and only if, $T \vDash F_{s,j} \sqsubseteq F_{t,k}$
- *O: Overflow*, occurs when the data target is subsumed by the source, if and only if, $T \vDash F_{t,j} \sqsubseteq F_{s,k}$
- *E: Exact flow*, occurs when the data source and target are equivalent, if and only if, $T \vDash F_{s,j} \equiv F_{t,k}$
- *N: No flow,* otherwise.

Figure 2 presents an example trace from our study reported in Section 5, produced using the Eddy toolset. There, the Flurry (F) collection requirements $F_{P17}$ and $F_{P24}$ trace to the transfer requirement $F_{P15}$; each requirement is represented as a node with arrows pointing in the direction of the inferred data flow. Nodes are annotated with the requirements expressed in

Eddy's syntax: "P" means permission, followed by the action verb and role values for *hasObject*, *hasSource* and *hasPurpose*. Requirement $F_{P15}$ does not specify the source from whom the data was collected, which we therefore assume to mean "anyone." Thus, the collection sources "end-user" and "application" are more specific than the transfer source "anyone," which appear as underflows (red arrows). The datum "device-id" in $F_{P15}$ is a subset of the data collected under $F_{P17}$ and $F_{P24}$, so the solid line arrow matching the datum is blue to show an overflow.

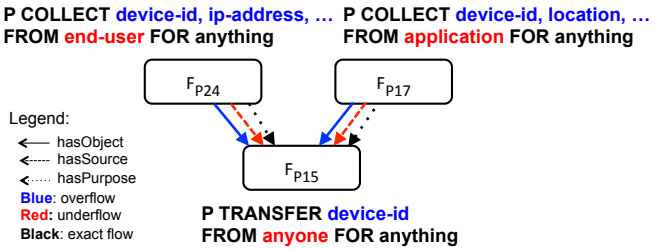


Fig. 2. Example data flow trace: solid lines represent data, dashed lines represent the data source, and dotted lines represent the purpose; these lines are marked ase overflows (blue), underflows (red); and exact flows (black)

Below, the collection requirement $p_1$ in formula (3) encodes requirement $F_{P24}$ from Figure 2, and $p_2$ in formula (4) encodes the transfer requirement for $F_{P15}$. In formula (2), we show that device-id is subsumed by the broader set of data collected under $F_{P24}$. Thus, the following are true:

(2) $T \vDash device\_id \sqsubseteq (device\_id \sqcup ip\_address \sqcup api\_key)$

(3) $T \vDash p_1 \equiv COLLECT \sqcap \exists hasObject.(device\_id \sqcup ip\_address \sqcup api\_key) \sqcap \exists hasSource.end\_user \sqcap \exists hasPurpose.Purpose$

(4) $T \vDash p_2 \equiv TRANSFER \sqcap \exists hasObject.device\_id \sqcap \exists hasSource.Actor \sqcap \exists hasTarget.Actor \sqcap \exists hasPurpose.Purpose$

Based on the subsumption relation entailed in formula (2), we can map the trace $(p_1, p_2) \longrightarrow (O, U, E)$ onto the three *Modes* for the roles *hasObject*, *hasSource* and *hasPurpose*, respectively. In general, tracing data flows allows an analyst to visualize dependencies between collection, use and transfer requirements. In this paper, we extend this formalization to enable traces across multi-party specifications, which requires a shared lexicon to unify terminology across specifications.

### *B. New Extensions to Privacy Specifications*

We now present extensions to privacy specifications to trace data flows across parties and check a specification for compliance with the three privacy principles.

*1) Tracing Multi-party Data Flows*. In multi-party data flows, we must trace from one actor's transfer permissions to another actor's collection permissions. For example, the Waze privacy policy describes permissions to collect and transfer personal information to and from social networks. However, Waze interoperates with multiple social networks, such as Facebook and Google+, who each provide this service to Waze. Each 3rd party has their own privacy policy that governs the flow of data to and from Waze. We formalize these relationships in a service map that assigns each party in the service relationship to one or more roles, and a dictionary that maps terminology from one specification to terminology in the other. Each specification can be written independently and thus developers need only align specifications based on their known service relationships, as opposed to the more difficult challenge of developing a universal ontology. To distinguish between actors and their specifications, we introduce the set *Agent* that contains unique identifiers for each actor in the domain of discourse. In addition, we introduce the mapping function $policy : Agent \longrightarrow Spec$ that maps agent $id \in Agent$ to their privacy requirements specifications in $Spec$. Below, we refer to agents $a_1, a_2 \in Agent$ and their respective privacy specifications in TBoxes $T_1$ and $T_2$.

**Definition 1**. A *service map* describes a service relationship between exactly two agents $a_1, a_2 \in Agent$. For each agent $a_i$, the map $M$ consists of one or more role pairs $(a_i, r_j) \in Agent \times Actor$ that maps a unique agent $a_i$ to one or more roles $r_j$ in the service relationship: for agent $a_1$, the role $r_j$ is in TBox $T_2$ of the other agent. For example, the role pair $(a_1, social\_network)$ maps the agent $a_1$ to the actor concept $social\_network \in T_2$ which defines agent $a_2$'s terminology. Each agent may have more than one role in a service relationship for some $j = 1 \ldots n$, but each of these roles is defined in the other agent's terminology. These roles are used to determine which transfer requirements to analyze based on the collecting agent's role, and vice versa.

**Definition 2**. Each service map contains a single *dictionary* $D$ that maps terminology from one agent's TBox $T_1$ to a second agent's TBox $T_2$. This mapping is used to compare actor, datum, and purpose descriptions in agent $a_1$'s source actions to corresponding descriptions in agent $a_2$'s target actions. The dictionary is expressed as a collection of axioms over concepts $t_i \, \varphi \, t_j$ wherein $(t_i, t_j) \in Actor \times Actor$, $Datum \times Datum$, $Purpose \times Purpose$, and $t_i \in T_1$, $t_j \in T_2$, and $\varphi$ is a DL equivalence operator ($\equiv$) or a subsumption operator ($\sqsubseteq, \sqsupseteq$).

For example, Waze's specification defines the concepts "unique device id" to refer to the mobile device identifier for Android or iOS devices, and "list of friends" refers to the list of contacts obtained from the Waze user's social networking service. To align the concepts with "device id" and "end user information" used in the Flurry specification, we construct a new TBox $T' = T_1 \cup T_2 \cup D$ such that (5) and (6) are true:

(5) $T' \vDash unique\_device\_id \equiv device\_id$

(6) $T' \vDash list\_of\_friends \sqsubseteq end\_user\_information$

**Definition 3**. A *multi-party trace* is a subset of requirements pairs $(reqt, reqc) \in Req_1 \times Req_2$ that map from a transfer requirement $reqt \in T_1$ to a collection requirement $reqc \in T_2$. For example, we may trace from permitted data transfers defined in first-party agent $a_1$'s TBox $T_1$ to permitted data collections in the counterparty agent $a_2$'s TBox $T_2$. To find multi-party traces from a first-party agent $a_1$ to their counterparty $a_2$ with respect to a service map $M$, we must identify all data transfers from $a_1$ to data recipient roles that the counterparty $a_2$ is expected to fill. We first express the DL union of counterparty $a_2$ roles $roles_2 = r_{2,1} \sqcup r_{2,2} \sqcup \ldots r_{2,n}$ for

all role pairs $(a_2, r_{2,i}), \ldots (a_2, r_{2,n}) \in M$. Next, we identify candidate transfer requirements $reqt_i \in Req_1$ where the recipient $hasTarget.r$ in the transfer requirement is an actor in the union of counterparty roles $r \sqsubseteq roles_2$. The set $C_{reqt}$ contains these candidate transfer requirements that are written in the form $reqt_i \equiv TRANSFER \sqcap \exists hasTarget.r \sqcap \exists R_{ti,2}.F_{ti,2} \sqcap \ldots \sqcap \exists R_{ti,n}.F_{ti,n}$ such that $R_{ti,2}, \ldots, R_{ti,n} \in Roles_1$, $F_{ti,2}, \ldots F_{ti,n}$ are fillers for those roles, and $r \sqsubseteq roles_2$. Conversely, we define the DL union of first-party roles $roles_1 = r_{1,1} \sqcup r_{1,2} \sqcup \ldots r_{1,n}$ for all role pairs $(a_1, r_{1,i}), \ldots (a_1, r_{1,n}) \in M$. The set of candidate collection requirements $C_{reqc} \subseteq Req_2$ consists of collection requirements written as $reqc_j \equiv COLLECT \sqcap \exists hasSource.s \sqcap \exists R_{cj,2}.F_{cj,2} \sqcap \ldots \sqcap \exists R_{cj,n}.F_{cj,n}$, such that $R_{cj,2}, \ldots, R_{cj,n} \in Roles_2$, $F_{cj,2}, \ldots F_{cj,n}$ are fillers for those roles, and $s \sqsubseteq roles_2$.

The candidate transfer requirements $C_{reqt}$ and collection requirements $C_{reqc}$ are only candidate members to multi-party traces, because the corresponding *hasTarget* and *hasSource* role fillers $r$ and $s$ are constrained by the corresponding agent roles in the service map. To identify the actual multi-party traces, however, we compare the remaining roles filers in candidate requirements pairs $(reqt_i, reqc_j) \in C_s \times C_t$ by constructing a new TBox $T_1' = T_1 \cup T_2 \cup D$. We compare the corresponding fillers $F_{i,2}, \ldots F_{i,n}$ and $F_{j,2}, \ldots F_{j,n}$ from requirements $reqt_i, reqc_j$, such that roles $R_{i,x} = R_{j,y}$. For our study, we are particularly interested in the roles *hasObject* and *hasPurpose*, but other roles may be included to constrain these traces. These comparisons yield one of the four exclusive modes (*underflow*, *overflow*, *exact flow* or *no flow*) described in Section 3A. For example, if $T' \vDash hasObject.F_{t3} \sqsubseteq hasObject.F_{c4}$ for all interpretations $\mathfrak{T}'$ that satisfy the TBox $T'$, then there is an underflow in the $hasObject$ role across requirements $reqt_3$ and $reqc_4$.

In Fig. 3, we present a multi-party trace from our case study. Within the Waze specification, the permitted collection $W_{P6}$ was traced to the permitted transfer $W_{P46}$; in particular, the Waze specification (and their privacy policy, for that matter) defines a Waze user's unique mobile device identifier to be a kind of personal information. Because formula (5) defines an equivalence axiom between the concepts *unique-device-id* and *device-id*, and Flurry is in the actor concept *ad-network*, $W_{P46}$ is traced to Flurry's permitted collection $F_{P1}$.

Definition 2 defines multi-party traces from transfers to collections, which can be generalized for any source and target actions in a specification, such as from uses to transfers, etc.

*2) Verifying the Limitation Principle.* The act of *repurposing* occurs when data that was collected for one purpose is used for a different purpose. Repurposing is a violation of the use limitation principle. The act of *over-collection* occurs when too much data is collected or when data is collected for more purposes than are needed. This is a violation of the collection limitation principle. We formalize these two principles into a general limitation principle; the purposes of all target permissions must be no greater than purposes for all source permissions with respect to the data of interest.

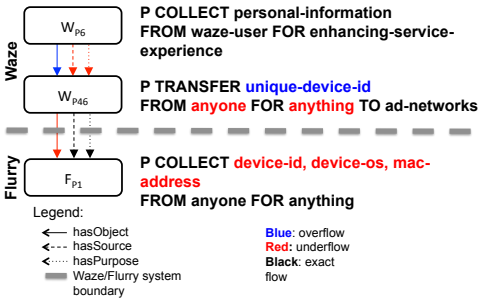


Fig. 3. Example multi-party data flow trace: solid lines represent data, dashed lines represent data sources, and dotted lines represent purposes; also shown are overflows (blue), underflows (red); and exact flows (black)

**Definition 4.** The limitation principle is verified in a specification $KB$ by first defining a set of limiting permissions $L_P$ and a new TBox $T'' = T \cup L_P$ for a conflict-free TBox $T$, and then proving that for all target permissions $t \in T_P$ there exists a limiting permission $l \in L_P$ such that $T'' \vDash t \sqsubseteq l$ for all interpretations $\mathfrak{T}''$ that satisfy the Tbox $T''$. The limiting permissions $L_P$ are derived from the set of source permissions $S_p$ by copying each source permission and replacing the source action $a_s$ with the target action $a_t$. We say the new TBox $T''$ is $t_a$-*limited*, if and only if, for any target permission $t \in T_P$, there exists a limiting permission $l \in L_P$ such that $T'' \vDash t \sqsubseteq l$; otherwise, the target permission $t$ violates the $t_a$-limitation principle, for some target action $t_a \in \{COLLECT, USE, TRANSFER\}$.

For example, we can define a *TRANSFER*-limitation principle, with the sets of permitted collections $S_p = \{s | s \in Req \wedge T \vDash s \sqsubseteq COLLECT \sqcap Permission\}$ and transfers $T_p = \{t | t \in Req \wedge T \vDash t \sqsubseteq TRANSFER \sqcap Permission\}$. The limiting permissions $L_p = \{l | s \in S_p \wedge (T \vDash s \equiv COLLECT \sqcap \exists R_1.F_2 \sqcap \ldots \sqcap \exists R_n.F_n) \rightarrow (T'' \vDash l \equiv TRANSFER \sqcap \exists R_1.F_2 \sqcap \ldots \sqcap \exists R_n.F_n)\}$. Consider the collection permission $W_{P6}$ from Figure 3, which appears in formula (7), below.

(7) $T \vDash p_6 \equiv COLLECT \sqcap \exists hasObject.personal\_info \sqcap \exists hasSource.waze\_user \sqcap \exists hasPurpose.enhancing\_service\_experience$

(8) $T'' \vDash l_6 \equiv TRANSFER \sqcap \exists hasObject.personal\_info \sqcap \exists hasSource.waze\_user \sqcap \exists hasTarget.Actor \sqcap \exists hasPurpose.enhancing\_service\_experience$

(9) $T \vDash p_{46} \equiv TRANSFER \sqcap \exists hasObject.unique\_device\_id \sqcap \exists hasSource.Actor \sqcap \exists hasTarget.ad\_networks \sqcap \exists hasPurpose.Anything$

From formula (7), we derive the limiting permission $l_6$ in formula (8) by replacing the source action *COLLECT* with the target action *TRANSFER* and by completing the missing role/value pair for *hasTarget*. The transfer permission $W_{P46}$ from Figure 3 appears in formula (9), above. While we know that *unique-device-id* is a kind of *personal-information* in Waze's specification, we cannot show that $T'' \vDash p_{46} \sqsubseteq l_6$, because the transfer purpose in $p_{46}$ exceeds the limiting purpose in $l_6$, e.g., "any purpose" subsumes "enhancing service experience". Assuming there is no other limiting permission that subsumes $p_{46}$, this transfer permission violates the *TRANSFER*-limitation principle.

## IV. Case Study Design

We evaluated the Eddy language extensions for multi-party tracing in an empirical case study [11, 25] using coding theory to increase construct and internal validity [19]. Coding is an interpretative, qualitative method in which multiple analysts assign codes from a coding frame to data. The coded data is statistically measured for agreement above chance, whereby sources of disagreement are used to disambiguate the coding frame. We examined the following five policies (the policies with asterisks were not formalized, explained below):

- Waze Privacy Policy, modified 30 May 2013
- Facebook API Developer Guidelines, revised 28 June 2013
- Amazon AWS Customer Agreement*, updated 15 March 2012
- Amazon AWS Privacy Policy*, acquired on 16 August 2013
- Amazon Privacy Policy*, updated 6 April 2012
- Flurry Privacy Policy, updated 9 July 2013

We removed Amazon's policies from the dataset because Amazon's contract language does not clearly cover end user data in their cloud services. We discussed this "gap" with a legal expert, who read all three policies and suggested that the absence of a specific privacy policy for end user data in Amazon's cloud services effectively allows Amazon to use the data as they wish within the confines of national or provincial laws. Therefore, we focused our analysis on Waze, Facebook and Flurry policies.

We sought to answer three research questions within the limits of our formalization of the three selected policies:

**RQ1**: What multi-party data flows exist across the selected policies?

**RQ2**: Does data repurposing or over collection occur within any of the policies, or across policies?

To answer these questions, we coded the three parties' policies using the coding methodology adapted from Saldana [19]. Our coding method has five steps, illustrated in Fig. 4: (1) we identify the policy statements that are *data requirements*, which specifically describe actions performed by humans or software on data; (2) for each data requirement, we assign a statement-level code to indicate whether the statement is either a *collection*, *use* or *transfer* of information based on a verb, and we assign phrase-level codes to identify the *actors*, *data types* and *purposes* relevant to each role (object, source, target and purpose); and (3) if the statement contains a definition or elaboration of a phrase-level concept, we code this phrase as either: a *refinement*, wherein one concept is refined by a more specific concept; an *abstraction*, wherein a list of concepts is described by a more general concept; or an *exclusion*, wherein a concept is excluded from another concept (i.e., assumed to be disjoint with). After each statement is analyzed, the analyst maps the coded text into Eddy syntax (step 4), which is compiled into DL using an automated parser tool (step 5).

In step 5, the parser reads the specification and compiles DL formulae. The terminology is compiled into equivalence, subsumption, and disjointness axioms (see formula A-B in Figure 5), and the requirements are compiled into requirements DL equivalence axioms (see formula C), subsumed by an appropriate modality (see formula D). In addition to the requirements specifications, we introduced service maps as an inter-lingua that maps concepts from one specification to another. Fig. 5 illustrates a service map from the case study between Waze and their advertising network, Flurry.

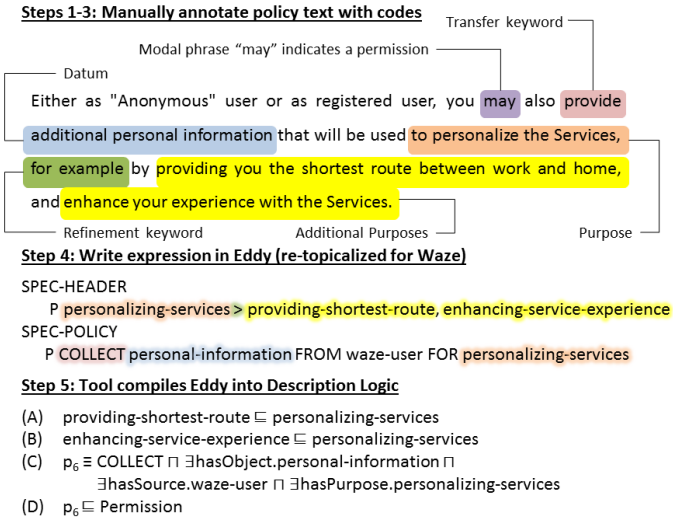


Fig. 4. Example policy statement coded as a data requirement

For example, in Fig. 5, Waze is called a "customer" in Flurry's specification, and Flurry is called an "ad-network" in Waze's specification. Each actor can have more than one role. Lines 3-12 describe definitions wherein the first actor's terms appear on the left-hand side of the definition operator ("<" means is subsumed by, ">" means subsumes, and "=" means is equivalent to) and the second actor's terms appear on the right-hand side. Some mappings may appear obvious, such as "age" (see line 7), whereas other terms must be inferred using domain knowledge: e.g., in advertising, frequency data such as the number of ads clicked is also called aggregate data (see line 3). In practice, we envision that requirements analysts would develop these terminologies with consultation from their legal departments, or they may reuse industry-wide terminologies from standards organizations. Similar to software APIs, we expect service mappings to be maintained over time, and could become relatively stable for services that do not frequently evolve their data practices.

```
1   NS1 http://localhost/waze-pp.owl customer
2   NS2 http://localhost/flurry-pp.owl ad-networks
3   D ads-clicked < aggregated-data
4   D ads-clicked = clicks
5   D ads-posted < aggregated-data
6   D ads-viewed < aggregated-data
7   D age = age
8   D list-of-friends < end-user-data
9   D location = location
10  D personally-identifiable-information < end-user-data
11  D profile-picture < end-user-data
12  D unique-device-id = device-id
```

Fig. 5. Example service map from Waze to Flurry; lines 1-2 set the terminological namespace for the first and second actor, followed by datum ("D") definitions for terms from the first actor (left side) that map to terms of the second actor (right side)

For the three coded policies reported in this paper, we measured inter-rater reliability at between .791 and .921 using

Cohen's Kappa. This is comparable to a separate study [8] using the same coding frame that yielded between .800 and .910 for Cohen's Kappa and also used two independent raters.

## V. Case Study Results

We now discuss our case study results. The first and third authors separately coded the Waze, Flurry and Facebook policies in three sessions. The average coding times required 135, 88 and 138 minutes, respectively. We coded 26-36% of the three policies, because large portions did not describe data actions covered by the coding frame. We computed Cohen's Kappa for all three policies and found inter-rater reliability to be .791, .921 and .925, which is a high degree of agreement above chance and includes agreement about which statements were excluded. Table I presents the total number of statements (*Total Stmts*) and number of data requirements (*Req'ts*), which are divided into the number of permissions (P), obligations (O), prohibitions (R), collections (C), uses (U) and transfers (T). Between 92-95% of Waze and Flurry privacy policies describe permissions; however, the Facebook policy contains more prohibitions (> 50%). This is because the Facebook policy regulates practices of connecting apps, such as Waze, and that includes which practices are not allowed.

TABLE I. Overview of Privacy Specification Composition

| Policy | Total Stmts | Data Req'ts | Modality | | | Action | | |
|---|---|---|---|---|---|---|---|---|
| | | | P | O | R | C | U | T |
| **Waze** | 150 | 65 | 60 | 0 | 5 | 13 | 18 | 34 |
| **Flurry** | 155 | 44 | 42 | 0 | 2 | 15 | 6 | 23 |
| **Facebook** | 136 | 55 | 24 | 1 | 30 | 13 | 24 | 18 |

Table II presents the number of definitions that were recorded explicitly from coding the policies (*Expl.*), and those that were later inferred to align terminology within specifications (*Infer.*). Inferences are needed because policy authors change terminology across adjacent sentences. All terms were explicitly stated in the policies, and only the formal relationships were inferred (e.g., "personal information" was inferred to be equivalent to "personal details"). Table II reports definitions by the number of subsumption (S), disjointness (D), and equivalence (E) axioms over actors (A), datum (D) and purposes (P).

TABLE II. Number and Types of Definitions, Explicit and Inferred

| Policy | Definitions | | Axioms | | | Concepts | | |
|---|---|---|---|---|---|---|---|---|
| | Expl. | Infer. | S | D | E | A | D | P |
| **Waze** | 41 | 37 | 70 | 2 | 6 | 13 | 46 | 19 |
| **Flurry** | 45 | 37 | 64 | 4 | 18 | 19 | 93 | 31 |
| **Facebook** | 45 | 8 | 50 | 0 | 3 | 12 | 73 | 28 |

Table III presents the number of alignments obtained by the coders, who created the service maps (described in Section 3B). Both coders created the service maps separately and then compared their results to reach consensus.

TABLE III. Number and Types of Mappings in Service Maps

| Service Map | Alignments | Axioms | | Concepts | | |
|---|---|---|---|---|---|---|
| | | S | E | A | D | P |
| **Waze-Facebook** | 519 | 498 | 21 | 14 | 432 | 73 |
| **Waze-Flurry** | 318 | 295 | 23 | 44 | 193 | 81 |

We now discuss the conflicts, multi-party data flow traces and limitation principle violations that we detected in our study.

### A. Potential Conflicts within Each Policy

Automated analysis detected potential conflicts within the Waze and Flurry privacy specifications that we classified into three categories: *ambiguous specifications conflicts* arising when the original policy statement lacks information that may have prevented the conflict, e.g., not stating a data transfer purpose implies "any purpose" that exceeds allowable transfer purposes; *electable permissions conflicts* that conflict with prohibitions in the policy prior to election, e.g., a policy may prohibit a particular collection, unless the user consents to the collection; and *direct conflicts* among practices that are unelectable and complete with respect to the a conflicting role specification. Developers should consider electable permissions conflicts as a source of *potential* conflict: if the system treats all data as elected for the permitted practice, this would be a violation of their policy. We now discuss conflicts due to ambiguous specifications and direct conflicts.

The automated analysis discovered 13 conflicts in Waze's policy that included eight ambiguous specification conflicts, three electable permission conflicts, and two direct conflicts. Waze's specification includes a prohibition ($W_{R1}$) that prevents transferring personally identifiable information (PII) to third parties for marketing purposes; this conflicts with permissions, (such as $W_{P45}$) below, that allow transfers to partners and service providers for unspecified purposes. This ambiguous specification conflict can be removed by modifying the purpose to explicitly exclude marketing purposes (see $W_{X45}$, below, where the disjoint operator "\" is read "anything except for marketing purposes").

```
W_R1: R TRANSFER PII TO third-party FOR marketing-purposes

W_P45: P TRANSFER PII TO partners, service-providers

W_X45: P TRANSFER PII TO partners, service-providers
       FOR anything \ marketing-purposes
```

These direct conflicts arise because Waze's specification prohibits transferring personal information to Waze users when the user is anonymous (see $W_{R0}$ below), while a separate permission allows posting user-uploaded information, along with other personal information, to be seen by Waze users (see $W_{P13}$). Among $W_{R0}$ and $W_{P13}$, there is a shared interpretation in the *hasObject* role value *personal-information* that excludes *driving-speed* and *time-joined-service* that is both prohibited by $W_{R0}$ and permitted by $W_{P13}$. Because this is a direct conflict, the specification author needs to carefully consider the impact of mitigating the conflict. Removing the prohibition may permit unintended data sharing, while remove the permission may break services that depend on this data flow. Alternatively, the purposes are unrestricted; thus, the author could introduce a specific class of purposes that are prohibited, while preserving a class of permitted purposes. If these two classes are logically disjoint, then the conflict would be eliminated.

```
W_R0: R TRANSFER personal-information \ driving-speed, time-
      joined-service TO waze-user

W_P13: P TRANSFER uploaded-information, personal-information
       TO waze-user
```

In the Flurry policy, the automated analysis discovered nine conflicts: eight conflicts due to ambiguous specifications and one direct conflict. The direct conflict results from Flurry permitting the transfer of Waze users' unique device id to advertisers who want to confirm conversion[1] via the Flurry ad network (see $F_{P27}$), despite transferring PII to non-affiliates being prohibited. Advertisers who buy ad space through Flurry, and who are not owned by Flurry's parent company, are not affiliate companies under Flurry's privacy policy.

```
F_R0: R TRANSFER PII to non-affiliated-companies

F_P27: P TRANSFER device-id TO advertisers FOR confirming-
     conversion-via-flurry-network
```

### B. *Tracing Multi-party Data Flows*

Based on Waze and Flurry's specifications and service maps, the automated analysis found 416 total traces within and across both policies, of which 73 traces were multi-party traces from Waze transfers to Flurry collections. This includes permissions and prohibitions; if a prohibition is paired with a permission, then this trace indicates a data supply or demand that is not reciprocated by the counterparty. Fig. 6 presents a subset of 12 from 416 traces, four of which are multi-party traces that cross the vertical dividing line between Waze's practices on the left and Flurry's practices on the right side.

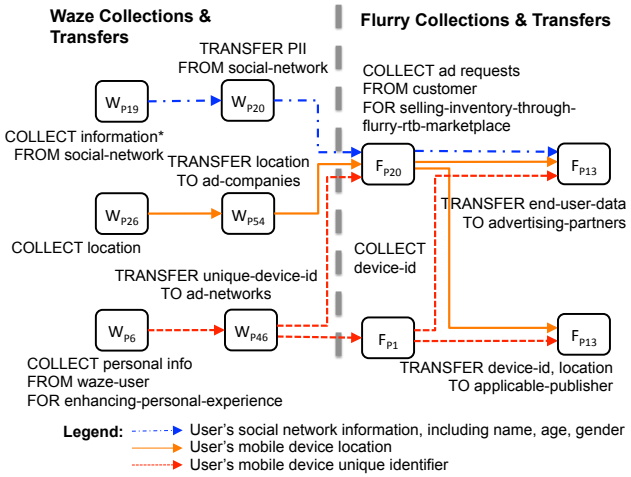


Fig. 6. Example multi-party data flow trace from Waze's privacy policy to Flurry's privacy policy: arrows point in the direction of flows, tracing Waze user data (blue), user location (orange) and users' unique device identifier (red) to Flurry as third-party advertisers

Demonstrated in the flow trace seen in Fig. 6, if a user connects to Waze via Facebook Login, the user's social network information, including their contact information, profile photo, list of friends, age, etc., may be transferred to third parties to support Waze services (see $W_{P19}$ and $W_{P20}$). This includes Flurry.com, who specifically collects ad requests that include end user data at Waze's discretion (called customer key/value pairs, see $F_{P20}$). This end user data is combined with the user's location and unique device identifier, all potentially sent to advertising partners (see $F_{P13}$). In addition, the Waze user's device id and location (collected and transferred via $W_{P6}$ and $W_{P46}$, respectively) are sent to applicable publishers that participate in Flurry's AppCircle mobile app advertising service (see $F_{P1}$ and $F_{P13}$). This uniquely identifying information is collected under the general purpose of "enhancing personal experience," and broadly re-purposed for any purpose by Waze and Flurry. If a Waze developer had not intended the user's social network information (including their list of friends) to be used for advertising purposes, they could state that intent in their privacy specification and rerun our analysis to detect a conflict with Flurry's policy which permits this data to be shared with advertisers.

### C. *Verifying the Three Privacy Principles*

We checked the Waze and Flurry specifications for compliance with the purpose specification principle, collection-, and use-limitation principles. The purpose specification principle requires companies to declare their data use purposes at the time of collection. Thus, collection requirements should all include stated purposes. The use and collection limitation principles aim to avoid repurposing and over-collection, which are two threats to privacy.

Table IV presents the ratio of missing purposes to total actions for collections, uses, and transfers that we automatically identified in the coded specifications. These missing purposes are ambiguities in the policies, which are preserved by the analyst when writing the policy statement using the Eddy language. The missing purposes are logically inferred to mean "any purpose" in the formal specification, which can lead to violations of limitation principles. In Table IV, collection and transfer statements were more likely (>50%) to omit purposes, whereas usage statements were less likely (<10%) to omit purposes.

TABLE IV. THE RATIO OF MISSING PURPOSES TO TOTAL ACTION TYPE FOR COLLECTIONS (C), USES (U) AND TRANSFERS (T)

| Policy | Missing Purposes | | |
|---|---|---|---|
| | Collections | Uses | Transfers |
| **Waze** | 6/13 (46%) | 1/18 (5%) | 23/30 (76%) |
| **Flurry** | 11/15 (73%) | 0/6 (0%) | 13/22 (59%) |
| **Facebook** | 4/8 (50%) | 1/10 (10%) | 5/7 (71%) |

The repurposing analysis examined Waze's data practices regarding three categories of user data: location, driving route, and personal information. The high frequency of missing purposes for collections and transfers reduces the risk of repurposing, because collection and transfer purposes that are omitted are interpreted as unrestricted (i.e., for "any purpose"). That said, the analysis detected that the Waze user's driving route was repurposed. The permitted collection $W_{P4}$, below. constrains the collection of *route* from Waze users for the purpose of providing services, whereas the specification also permits the use ($W_{P8}$) and transfer ($W_{P7}$) of *driving-route* for any purpose. This repurposing from "providing services" to "any purpose" violates use- and transfer-limitation principles, and may be corrected by such purposes to be subsumed by collection purposes.

[1] In advertising, a *conversion* occurs when a consumer observes an ad and proceeds to make a purchase from the featured seller, including a purchase of the advertised product. Conversion may be measured by linking the ad-observation event to the consumer's location at the seller's store or to a known purchase in the consumer's purchase history.

$W_{P4}$: **P COLLECT location, route FROM waze-user FOR providing-services**

$W_{P7}$: **P TRANSFER avatar, distance, driving-route,…, waze-rank**

$W_{P8}$: **P USE avatar, distance, driving-route,…, waze-rank**

Recall that over-collection violates the collection limitation principle, because more information is collected than needed. We detect over collection by testing whether collection purposes are subsumed by use and transfer purposes. When specifications have a high frequency of unrestricted transfer purposes (as shown in Table IV), the risk of over collection is reduced, since developers can argue that broad collection purposes are needed to support broad transfer purposes, regardless of use. However, if we focus on use specifications and omit transfers from analysis, then we find over collection with respect to explicit uses. For example, $W_{P10}$ and $W_{P26}$, below, permit collection of location for any purpose, whereas the permitted use $W_{P18}$ restricts usage to identifying attractions and shops, presumably for notifying the user. Because there are no other declared usage purposes for location, this can be reported as an over-collection in the absence of transfers.

$W_{P10}$: **P COLLECT location FROM third-party-service, waze-user**

$W_{P26}$: **P COLLECT ads-clicked, ads-viewed, geographic-location,…, web-pages-visited**

$W_{P18}$: **P USE location, route-information FOR identifying-attractions, identifying-shops**

Our analysis uncovered two privacy specification design patterns whereby specification authors can bypass the limitation principles, i.e., they can comply with the principles by writing policies that violate the general spirit of the principles. The first *purpose hoisting* pattern describes permitted actions with restricted purposes (e.g., using route information in $W_{P18}$). These actions comply with purpose specification, and under the limitation principle serve to constrain any target actions. Next, the specification author describes the same permitted action with a broad purpose (e.g., $W_{P8}$ - assuming *driving-route* is a kind of *route-information*). This last action subsumes the other purposes and thus "hoists" the information from a restricted to a more general purpose, and thus reduces the risk of limitation principle violations. Purpose hoisting allows a policy author to be both specific and flexible in specifying their access to data.

The second *unrestricted cross-flows* pattern describes the source and target action purposes in general terms to offer greater design flexibility and potentially the least privacy. We identified instances of this pattern in Waze's specification. For example, collections and transfers of personal information to and from social networks are unrestricted by purpose. Collection of Waze users' location is unrestricted through transfers to other Waze users, non-Waze users and advertising companies. In this regard, Waze has considerable flexibility in how they share data with social networks, or how they use location. Such flexibility may benefit companies who evolve their business models, but it may introduce privacy risks.

### *D. Tool Support for Scaling Multi-Party Compositions*

The Java-based Eddy language parser and compiler produce specifications expressed in Web Ontology Language (OWL) DL, which is then analyzed using a theorem prover. Prior simulations [10] concluded that the HermiT Reasoner v1.3.4 produces low-coefficient exponential asymptotic performance with respect to policy size and number of conflicts when analyzing single party policy specifications. This suggested some measure of scalability toward policy specifications involving multiple parties. We replicate and present these results and asymptotic characteristics in Fig. 7, in which policies have a fixed number of data elements, and increasing permission and prohibitions (up to 80).

In our new analysis, we evaluate policies larger than those in our case study. The generated policies contain: 400 statements (100 collection, use, and transfer permissions and 100 collection prohibitions); a fixed number of actors and purposes (10 actors and 10 purposes, each with 10 children subsumed by their parent); and up to 100 data flows between parties. This policy configuration is more than double the size of policies seen in Section V. The data element count was varied from 1-52 in the first layer of each policy's ontology, with 4 layers beneath based on our analysis of a combined 26-policy ontology. Finally, we performed 20 repetitions per datum and averaged record times. Figure 8 presents the results, showing the exponential characteristics of the time complexity for the parsing and analysis. Even with the max 52 data elements, the average time required for parsing and analysis was under 8 minutes (475 seconds). Tests were performed on commercial off-the-shelf hardware using 1st generation Intel i7 quad-core processors running at 2.9GHz with 6GB of RAM (all dedicated to the Java Virtual Machine), characteristic of a typical workstation from c. 2010.

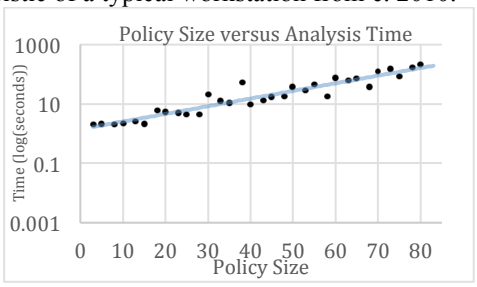


Fig. 7. Logarithmic plot of the number of policy statements versus average 20-run automated analysis time (seconds) for conflict detection.

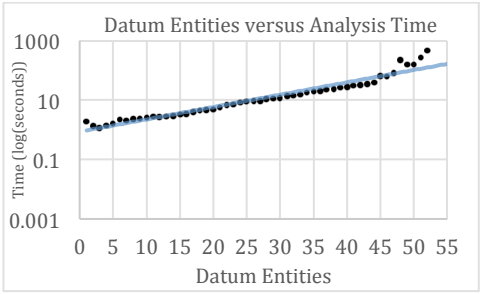


Fig. 8. Logarithmic plot of the number of top-level datum entities in policy versus average 20-run automated analysis time (seconds) for flow tracing.

We made the Eddy source code, API documentation and an online demo available to the public.[2]

[2] http://cmu-relab.github.io/eddy

## VI. Threats to Validity

We designed our case study evaluation to reduce threats to validity. *Construct validity* concerns whether measures actually measure the construct of interest [25]. We chose privacy policies as a data source from which to develop and evaluate our approach. To increase construct validity, we use multiple coders to achieve high agreement (Cohen's Kappa between .791 and .921) regarding the encoding of the privacy policies in Eddy. However, we know that policies summarize numerous data flows across systems as evidenced by the magnitude of results in prior information flow analysis studies, and developers may have a more nuanced view of their data practices due to their familiarity with the system implementation and architecture.

*Internal validity* refers to whether the conclusions drawn from the data are valid [25]. To reduce threats to internal validity, we extensively documented our encoding method, employed research notebooks, and replicated steps 1-3 of our method in a prior study [8] and the study presented herein.

*External validity* refers to the extent to which the results of this study can be generalized to other situations [25]. We examined only three policies that contain data requirements. In addition, the data actions in our language may not be complete with respect to developers needs to express their requirements. Also, our experience shows companies use different policy formats and levels of detail in their data practice descriptions, and this can affect the dimensionality of the policies and service maps. To reduce this threat, we evaluated our approach in a simulation with policies twice the size of our case study.

## VII. Related Work

As the volume of sensitive personal information available to software developers increases, privacy will receive more attention in software engineering research. Spiekermann and Cranor framed privacy as either *privacy-by-policy*, which concerns using activities such as privacy notice and consumer consent to improve privacy, or *privacy-by-architecture*, which focuses on data minimization and activities within the control of software [20]. We now review related work in software requirements engineering, including findings to characterize privacy requirements and methods to surface privacy threats and their mitigations.

Requirements engineering research has demonstrated that policies and regulations are rich sources of privacy requirements. Using grounded analysis, Antón et al. discovered a taxonomy of requirements in privacy policies, including dichotomous categories for types of privacy protections and vulnerabilities [2]. Subsequent work by Breaux et al. has led to formalizing privacy policy goals to detect conflicts [6]. Based on the earlier work by Wan and Singh [23], Young and Antón extracted policy commitments from privacy policies and terms-of-use agreements to find data collection and use requirements [26]. In addition to different company policies, Breaux and Antón mined U.S. health privacy law to discover 300 data access requirements and new techniques for requirements prioritizing with exceptions [5]. We extended this early work [5, 6] herein to trace data flows.

Early work to model privacy in requirements includes Liu et al., who extended the i* framework to reason about attackers and malicious intent [15]. More recently, Omoronyia et al. describe a method to model adaptive privacy that incorporates user context and use histories to detect emerging privacy threats [17]. They show how to compute the utility of disclosure by comparing the threat severity with the disclosure benefit. Tun et al. introduce privacy arguments based on selective disclosure or norms to analyze user privacy preferences for mobile apps that can change based on context [21]. The arguments are formalized in Event Calculus to reason about satisfaction. Salas and Krishnan show how to generate privacy requirements test cases from models [18].

In security research, privacy models exist to reason about data access, such as the HIPAA Privacy Rule [4, 17] and Privacy Act [12]. Barth et al. encoded regulations as messages passed between actors using norms (e.g., permitted and prohibited actions) [4], which is similar to Aucher et al. [1]. Hanson et al. introduce data purpose algebra to calculate the set of restrictions under which data may be used [12], which is similar to the use limitation principle. May encoded privacy regulations in Promela and used the Spin model checker to identify potential conflicts [16]. This prior work includes the use of Temporal Logics to reason about pre- and post-conditions on access, but they do not include the concept hierarchies needed to reason about data and purpose specifications. The Web Ontology Language (OWL) is used to express policies as permissions, obligations and prohibitions, including concept hierarchies [8, 14]. The full OWL, which these approaches use, is undecidable.[3] Work by Uszok et al. [22], however, uses an unpublished algorithm to identify conflicts; an approach that may be decidable, but is difficult to reproduce. Recently, Breaux and Rao extended this prior work by reducing conflict detection in privacy requirements to Description Logic (DL) satisfiability, which is decidable and PSPACE-complete for the ALC family of DL [9].

## VIII. Discussion and Summary

In this paper, we present extensions to the privacy requirements specification language, called Eddy, to model multi-party data flows and to verify the purpose specification and collection and use limitation principles. We operationalized these principles as a general limitation principle that can be used to restrict the actionable purposes from one action to another (e.g., collect-use, use-collect, collect-transfer, etc.) The principles are intended to provide strong, well-known guarantees that systems minimize personal information use and disclosure with respect to stated collection purposes, which supports engineering privacy [20]. To our knowledge, our formalization is the first to comprehensively enforce these principles in data flow specifications. In addition, we identified several challenges for future work. First, the Eddy language is aimed at supporting developers who want to check for conflicts between permitted and prohibited data practices, while detailing the exact logical interpretation that leads to the conflict. As shown in our case

[3] http://www.w3.org/TR/owl-ref/

study results, refining the requirements specification and removing ambiguity can remove some conflicts and violations of the limitation principles. Based on these results, we envision proof assistants that help analysts walk through potential conflicts and suggest alternative strategies for de-conflicting specifications. Automated suggestions may include further restricting permissions, or relaxing prohibitions.

The Eddy language is designed so that developers can independently state their design intent up to the boundaries of their system (i.e., what information they collect, use and transfer), while maintaining limited knowledge of the specific practices of their third-party service providers. The multi-party data flow tracing that we introduce in this paper aims to support developers who subsequently check whether third party uses conform to use limitations. We recognize the challenge of proving properties over third-party specifications while third parties protect their specifications as trade secrets, e.g., concealing the names of their service providers and specific internal business purposes. However, they could use a trusted third party to perform these checks.

We envision scenarios wherein developers realize new opportunities to use data in support of developing new system features. Through multi-party data flow traces, developers could identify data sources available in their system context or from third-party services and then, check whether existing usage purposes permit their envisioned data use. If the purposes do not allow such uses, developers could request that specifications be altered to support such requests. This may include allowing some users to elect to participate in the new feature (i.e., opt-in). By aligning the formal privacy specifications proposed herein with system feature exploration, we envision controlled personalization wherein user privacy preferences are not violated en masse, but are selectively relaxed for users who are willing to participate by coordinating developer design intent with user preferences.

Finally, we are studying crowdsourcing as a means to scale applications of the Eddy language to privacy policies [9].

## ACKNOWLEDGEMENTS

We thank Joel Reidenberg for his support and legal analysis on our early results and drafts of this paper. Supported by NSF Award #1330596, ONR Award #N002441410028 and the National Security Agency.

## REFERENCES

[1] G. Aucher, G. Boella, L. van der Torre. “Privacy policies with modal logic: a dynamic turn,” *Lect. Notes. Comp. Sci.* 6181: 196-213, 2010.

[2] A.I. Antón, J.B. Earp, “A requirements taxonomy for reducing web site privacy vulnerabilities,” *Req’ts Engr. J.*, 9(3):169-185, 2004.

[3] F. Baader, D. Calvenese, D. McGuiness (eds.), *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[4] A. Barth, A. Datta, J.C. Mitchell, H. Nissenbaum, “Privacy and contextual integrity: framework and applications,” *IEEE Symp. on Sec. & Priv.*, 2006, pp. 184-198.

[5] T.D. Breaux, A.I. Anton. “Analyzing regulatory rules for privacy and security requirements,” *IEEE Trans. Soft. Engr.*, 34(1): 5-20, 2008.

[6] T.D. Breaux, A.I. Antón, J. Doyle, “Semantic parameterization: a conceptual modeling process for domain descriptions.” *ACM Trans. Soft. Engr. Method*., 18(2): Article 5, 2009.

[7] T.D. Breaux, D.L. Baumer, “Legally ‘Reasonable’ Security Requirements: A 10-year FTC Retrospective.” *Computers & Security*, 30(4):178-193. 2011.

[8] 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.

[9] T.D. Breaux, F. Schaub, “Scaling requirements extraction to the crowd: experiments on privacy policies.” *IEEE 22nd Int’l Req’ts Engr. Conf. (RE'14)*, pp. 163-172, 2014.

[10] T.D. Breaux, A. Rao. “Formal analysis of privacy requirements specifications for multi-tier applications,” *IEEE 21st Int’l Req’ts Engr. Conf.,* pp. 14-23, Jul. 2013.

[11] J.W. Creswell. *Research Design: Qualitative, Quantitative and Mixed Methods Approaches, 2nd ed.* Sage Publications, 2003.

[12] C. Hanson, T. Berners-Lee, L. Kagal, G.J. Sussman, D. Weitzner, “Data-purpose algebra: modeling data usage policies.” *8th IEEE Work. Pol. Dist. Sys. & Nets.*, 2007, pp. 173-177.

[13] J.F. Horty. “Deontic logic as founded in non-monotonic logic.” *Annals of Math. & Art. Intel.*, 9: 69-91, 1993.

[14] M. Kahmer, M. Gilliot, G. Muller, "Automating Privacy Compliance with ExPDT." *IEEE 10th Conf. E-Com. Tech.*, pp. 87-94, 2008

[15] L. Liu, E. Yu, J. Mylopoulos. "Security and privacy requirements analysis within a social setting," *IEEE 11th Int'l Req'ts Engr. Conf.*, pp. 151-161, 2003.

[16] M.J. May, *Privacy APIs: Formal Models for Analyzing Legal and Privacy Requirements*, Ph.D. Thesis, U. of Pennsylvania, 2008.

[17] I. Omoronyia, L Cavallaro, M. Salehie, L. Pasquale, B. Nuseibeh. “Engineering adaptive privacy: on the role of privacy awareness requirements,” *IEEE 35th Int’l Conf. Soft. Engr*., pp. 632-641, 2013.

[18] P.P. Salas, P. Krishnan. "Testing privacy policies using models," *IEEE 6th Int'l Conf. Soft. Engr. Frml. Mthd*. pp. 117-126, 2008.

[19] J. Saldaña, *The Coding Manual for Qualitative Researchers*, Sage Pubs. 2012.

[20] S. Spiekermann, L.F. Cranor. "Engineering Privacy," *IEEE Trans. Soft. Engr.,* 35(1): 67-92, 2008.

[21] T.T. Tun, A.K. Bandara, B.A. Price, Y. Yu, C. Haley, I. Omoronyia, B. Nuseibeh. “Privacy arguments: Analysing selective disclosure requirements for mobile applications,” *IEEE 20th Int'l Req'ts Engr. Conf*., pp. 131-140, 2012.

[22] A. Uszok, J.M. Bradshaw, J. Lott, M. Breedy, L. Bunch. “New developments in ontology-based policy management: increasing the practicality and comprehensiveness of KAoS.” *IEEE Work. on Pol. Dist. Sys. & Nets*., pp. 145-152, 2008.

[23] F. Wan, M.P. Singh. “Formalizing and achieving multiparty agreements via commitments.” *Auto. Agents & Multi-Agent Sys.*, pp. 770–777, 2005.

[24] Y.S. Van Der Sipe, W. Maalej, “On lawful disclosure of personal user data: What should app developers do?” *IEEE 7th Int’l W’shp on Req’ts Engr. & Law,* pp. 25-34, 2014.

[25] R. Yin, *Case Study Research: Design and Methods*, 4th ed. Sage Pubs. 2008.

[26] J. Young. “Commitment analysis to operationalize software requirements from privacy policies.” *Req’ts Engr. J.*, 16:33-46, 2011