# DataTags, Data Handling Policy Spaces and the Tags Language

Michael Bar-Sinai
Computer Science Dept.
Ben-Gurion University of the Negev
Be'er-Sheva, Israel

Latanya Sweeney
Data Privacy Lab
Harvard University
Cambridge, MA

Mercè Crosas
Institute for Quantitative Social Science
Harvard University
Cambridge, MA

***Abstract*—Widespread sharing of scientific datasets holds great promise for new scientific discoveries and great risks for personal privacy. Dataset handling policies play the critical role of balancing privacy risks and scientific value. We propose an extensible, formal, theoretical model for dataset handling policies. We define binary operators for policy composition and for comparing policy strictness, such that propositions like "this policy is stricter than that policy" can be formally phrased. Using this model, The policies are described in a machine-executable and human-readable way. We further present the Tags programming language and toolset, created especially for working with the proposed model. Tags allows composing interactive, friendly questionnaires which, when given a dataset, can suggest a data handling policy that follows legal and technical guidelines. Currently, creating such a policy is a manual process requiring access to legal and technical experts, which are not always available. We present some of Tags' tools, such as interview systems, visualizers, development environment, and questionnaire inspectors. Finally, we discuss methodologies for questionnaire development. Data for this paper include a questionnaire for suggesting a HIPAA compliant data handling policy, and formal description of the set of data tags proposed by the authors in a recent paper.**

## I. Introduction

Wide dissemination of datasets holds great promises for science — findings can be corroborated, research cost reduced through data reuse, and new studies are made possible by combining existing data into new datasets, to name a few. But wide dissemination of datasets also poses risks to the data subjects. Privacy of human subject has to be respected. Precise locations of endangered species populations, rare minerals, or ancient ruins should not be easily available to poachers, illegal miners and tomb raiders. Many laws, regulations and best-practices were formed in order to balance the positive and negative potential of data sharing. In the US alone, there are more than two thousand rules and regulations governing data sharing [15]. To share data while respecting legislation, the scientific community has resorted to specialized data repositories, e.g. repositories specifically designed for medical data. While legally sound, this approach leads to fractured data storage infrastructure, and leaves datasets that contain data governed by certain law combinations without a repository.

In [16], the authors propose the concept of *datatags* as a way of ensuring that the handling of, and the access requirements to a dataset are commensurate with the risks of harm it poses. When deposited in a datatags-compliant data repository, a dataset is associated with a *datatag* which defines a machine-actionable policy under which the dataset should be handled. This ensures that the data repository handles the dataset properly, from a legal and contractual standpoint. By limiting the amount of datatags to a few well-defined choices, managing, reasoning about, and implementing robust datatags-compliant repositories becomes easier. Finally, a sample set of datatags is suggested (see Figure 1).

| *Tag Type* | *Description* | *Security Features* | *Access Credentials* |
|---|---|---|---|
| Blue | Public | Clear storage, Clear transmit | Open |
| Green | Controlled public | Clear storage, Clear transmit | Email- or OAuth Verified Registration |
| Yellow | Accountable | Clear storage, Encrypted transmit | Password, Registered, Approval, Click-through DUA |
| Orange | More accountable | Encrypted storage, Encrypted transmit | Password, Registered, Approval, Signed DUA |
| Red | Fully accountable | Encrypted storage, Encrypted transmit | Two-factor authentication, Approval, Signed DUA |
| Crimson | Maximally restricted | Multi-encrypted storage, Encrypted transmit | Two-factor authentication, Approval, Signed DUA |

Fig. 1. Blue to Crimson model set of datatags, proposed in [16]. The Blue tag is appropriate for datasets that pose no risks. As the risk level increases, so does the required access credentials, the security imposed by the handling requirements, and the strictness of the DUA terms and execution.

Datatags-compliant repositories will help automate open science, and facilitate data sharing [13]. Furthermore, since Datatags-based systems can predict the legal and technological requirements for the collection and handling of a dataset, they can also serve other use-cases, such as IRBs and research design.

Once a set of datatags has been defined, two challenges need to be addressed. First, a tag has to be formally described, in a way that allows formal reasoning about the data handling policies the tags imply, and clear transition into implementation. Second, a mechanism to help users match a dataset with the datatag most appropriate for it has to be created. Such matching requires familiarity with the dataset's history and collection methods, and legal and technological expertise — the latter two are not always readily available to researchers. Enforcing the policy detailed by the datatags is a challenge in its own right, is not in the scope of this paper.

In order to be useful, a datatag matching system should have

the following traits:

- *Algorithmic* Require input from the data depositor only.
- *Deterministic* Given the same input the system should yield the same results (assuming no legislation or technological changes).
- *User friendly* Deciding on the datatag of a dataset, and hence on the data handling policy applied to handle it, has ethical and legal implications. Thus, data depositors need to be well informed during the process, and user errors should be minimized.
- *Maintainable* The system captures non-trivial legal and technological knowledge, that has to be composed and maintained by human domain experts.
- *Assume reasonable user knowledge* Data depositors should not be expected to have legal or technological expertise. They are expected to be familiar with the way the dataset in question was collected.

This paper will propose an answer to both challenges, defining data handling policies as points in a mutli-dimensional space, and creating a datatag decision aid using a decision graph. The rest of this paper is organized as follows: Section II proposes a formal description of data handling policies, and uses this description to define datatags. Section III describes Tags, our open source datatags system. Section IV presents an example questionnaire. Section V mentions some of Tags' tools and briefly discusses methodologies for using it. Sections VI and VII discuss related and future work.

## II. Policies as Spaces

A given a data handling policy can be described as composed from many independent aspects. For example, the requirement to encrypt data 'at rest'[1] is independent of the requirement to have data users sign a data use agreement (DUA), or even from the requirement to encrypt the data on transmission. Granted, a policy that requires a signed DUA and an encrypted storage is likely to also require encrypted transmission. It is possible, however, to create a policy that requires encrypted storage and signed DUA, but allows clear transmission. It is therefore useful to allow description of such policies, if only to create an algorithm that warn against them.

Each of the data handling policy aspects has multiple possible requirements. For example, a policy might allow accepting a data use agreement in an implied manner, require using a click-through agreement process, or require a signed document. It might allow storage to be in the clear, encrypted, or even multi-encrypted[2]. These possible requirements can be ordered from lenient to strict.

In cases where the possible requirements of a data handling policy aspect cannot be ordered, it is possible to either generalize them, or split the aspect. For example, verifying user registration via an email address or an OAuth account is equivalent, as both demonstrate an account on an external system. Thus, as possible requirements for `VerifyRegistration`, `email` is no stricter than `oAuth`, nor is `oAuth` stricter than `email`. The solution is to replace them with a more general `usingExternalSystem` possible requirement. The actual choice between email, OAuth, or other equivalent technologies is then left to data repository system implementors.

An example for an overly broad aspect definition is `Encryption`. As there are two possible aspects to encrypt — storage and transmission — the possible requirements `transferOnly` and `storageOnly` have no strict ordering. Splitting the `Encryption` aspect to `StorageEncryption` and `TransmissionEncryption` solves this issue.

We propose viewing the independent aspects of data handling policies as orthogonal axes of an ordinal space, where the possible requirements of a given aspect are the coordinates along its axes. We call this space "Data Handling Policy Space" (or DHP space for short). Viewed like this, data handling policies become points in a DHP space. See Fig. 2 for a 2D example.

More formally, given independent aspects $a_1, a_2, \ldots a_n$, where each aspect $a_i$ has $k_i$ possible requirements $r_i[1], r_i[2], \ldots r_i[k_i]$, a data handling policy $P$ is a point in an $n$-dimensional data handling policy space, and is specified by the n-tuple:

$$P = \langle r_1[l_1], r_2[l_2], \ldots r_n[l_n] \rangle$$

Where $1 \leq l_i \leq k_i$. Aspect order is arbitrary, but of course has to be consistent once determined.

### A. Relations Over Policies

Building on the above description of a data handling policy, we now define equality and partial order, based on strictness.

**Equality** Given policies $P = \langle p_1, \ldots, p_n \rangle$ and $Q = \langle q_1, \ldots, q_n \rangle$, defined in the same DHP space, we say that $P$ is *equal to* $Q$ iff all their requirements are equal. Formally:

$$P = Q \iff \forall i \in [1..n].p_i = q_i$$

**Strictness** Given DHP space $S$ defined over $n$ aspects, and policies $P = \langle p_1, \ldots, p_n \rangle$ and $Q = \langle q_1, \ldots, q_n \rangle$ defined in $S$, we say that $P$ is *stricter than* $Q$ iff along all aspects of $S$, $P$ has requirements that are equal or stricter than $Q$'s, and on at least one aspect that value is strictly stricter. Formally:

$$P > Q \iff \forall i \in [1..n].p_i \geq q_i \\ \wedge \exists i \in [1..n].p_i \gneq q_i$$

For any DHP space with more than a single aspect, *strictness* is a partial order relation.

**Derived** Using equality and strictness, the relations *more lenient* ($<$), *stricter than or equal to*

[1]Encrypting the data before writing it to disk. This protects the data from being read be parties who have access to the file system, either over a network or physically. At rest encryption has performance downsides, and runs the risk of losing the data if the encryption key is lost.

[2]Requires multiple keys to decrypt. This type of requirement may prevent the dataset repository and its staff from reading the data in a dataset.

($\geq$), and *more lenient than or equal to* ($\leq$) are intuitively defined.

Composition Given DHP space $S$ defined over $n$ aspects, and policies $P = \langle p_1, \ldots, p_n \rangle$ and $Q = \langle q_1, \ldots, q_n \rangle$ defined in $S$, $P$ and $Q$ can be composed to a new policy, by selecting the stricter requirement for each aspect. Formally:

$$P \oplus Q \triangleq \langle \max(p_1, q_1), \ldots, \max(p_n, q_n) \rangle$$

Where $\max(f_i[a], f_i[b])$ is defined as $f_i[a]$ if $f_i[a] > f_i[b]$, and as $f_i[b]$ otherwise. Note that by definition $P \oplus Q \geq Q$ and $P \oplus Q \geq P$.

### B. Compliance and Support Sub Spaces

Security and access restrictions are not breached by overdoing. For example, if a dataset is allowed to be transmitted in the clear, it is also acceptable to encrypt it during transmissions. Formally, if a dataset is required to use policy $P$, a dataset repository may handle it according to any policy $Q$, when $P \leq Q$. Thus, each policy $P$ in DHP space $S$ defines two sub-spaces, demonstrated in Figure 2:

Compliance Space All policies that do not breach $P$:

$$compliance(P) \triangleq \{Q \in S | P \leq Q\}$$

Support Space All policies that $P$ does not breach:

$$support(P) \triangleq \{Q \in S | Q \leq P\}$$

A dataset requiring policy $P_1$ can be handled using any policy in $compliance(P_1)$, without breaching any laws. A data repository implementing policy $P_2$ can properly handle any dataset which requires a policy is in $support(P_2)$. In order to decide whether dataset $D$, requiring policy $P_D$, can be stored in repository $R$ under policy $P_R$, it is enough to require that $compliance(P_D) \cap support(P_R) \neq \emptyset$. When this assertion holds, we can say that *R can handle D using policy P*, writing $R \vDash_P D$. In the interest of open science, it is better to use the most lenient policy within that intersection.

Statistical disclosure limitation tools and sharing a differentially private version of the dataset [8], [9] can be viewed as moving the dataset to a more lenient point in DHP, at the expense of some of its aspects, such as its level of detail.

### C. DataTags and DHP

Under the Data Handling Policy Space model, DataTags are a set of points in a defined DHP space, totally ordered by strictness. The compliance and support spaces of the tags are hierarchically contained. For the set of tags defined in [16], $compliance(blue)$ is the outermost compliance space, which serves the intuition that handling an open access dataset as if it required all its users to be fully accountable (*red* tag) will not create a security breach (it would, however, unnecessarily complicate the dataset handling). The outermost support space is $support(crimson)$, serving the complementary intuition.

The code addendum for this paper [5], contains the datatag set proposed in [16] phrased using this approach.

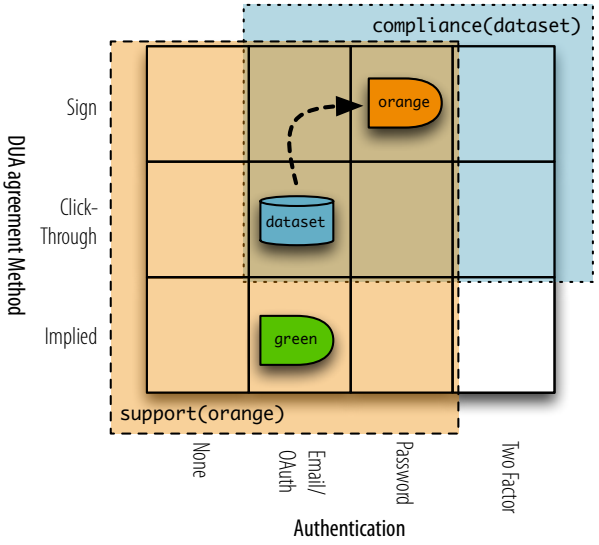


Fig. 2. Given a datatags repository $R$ supporting the Green and Orange tags defined in [16], and a dataset $D$ requiring a click-through DUA and an Email authentication, $D$ could be stored in $R$ under the Orange tag. This can be written as $R \vDash_{orange} D$

## III. Tags, a Graph-Based DataTagging System

We now present Tags, domain-specific programming language for defining data handling policy spaces, and creating friendly interactive questionnaires, that helps users arrive at a proper policy for a given dataset. We will also look at some of the tools available for Tags programs. Tags is an open source project under active development, so this paper covers only its basic concepts. For an up to date information, we refer the reader to the project website [4].

The DataTags concept serves multiple use cases, such as individual researchers, research groups, IRBs, and scientific and commercial data repositories [16]. Thus, the Tags parser and runtime engine are available in an embeddable Java library. This Library is currently embedded in two systems: CliRunner, a Java console application which serves as a development aid, and TaggingServer, a web application for conducting interviews, written in Scala.

A Tags program is called a *questionnaire*. An *interview* is the act of a user interactively answering questions from a questionnaire. Questionnaires are modeled after an interview with an expert, where the expert asks questions using a language the data depositor can understand, while writing comments in her notebook using professional, accurate, language. A Tags questionnaire is composed of two components: a *tag space* defining the professional language, and a *decision graph*, defining the questions and the flow of the interview. Tag space is similar to data handling policy space, but contains additional axes for auxiliary assertions, such as regulatory information and internal program variables. These additional assertions can be used later, e.g. for automatic customization of data use agreements. Different decision graphs may refer to the same tag space. Consequently, tag spaces may be used to define

standards that data repositories can comply with.

Tags was designed to be used by domain experts rather than programers. While it does require its users to adhere to formal syntax, it is a high-level language, and does not require handling memory allocation or read/write operations. To better serve the questionnaire development process, Tags supports top-down design, by allowing parts of the questionnaire to be marked as `TODO`.

### A. Tag Space

A direct description of multi-dimensional space — namely, a list of dimensions and the possible values along them — may be hard to work with, especially as the number of described aspects increases. Furthermore, such list does not allow for logical grouping, and hinders collaboration, both are crucial for engineering a questionnaire. Thus, Tags describes tag spaces using named slots, and defines a simple algorithm for generating a list of dimensions from a set of slots. Tags supports four types of slots:

*1) Atomic Slot:* A slot containing a single atomic value. Values defined for this type of slot are totally ordered. Slots of this type are defined using the `one of` keyword.

```
Storage: one of clear, encrypt, multiEncrypt.
```

*2) Aggregate Slot:* A slot that contains a set of values of its item type. Aggregate slots can be used to logically group multiple binary dimensions. They are defined using the `some of` keyword.

```
ProtectedDataSubjects: some of livingPersons,
  deceasedPersons, endangeredSpecies, rareMinerals.
```

*3) Compound Slot:* Slots of this type consist of other slots, referred to as sub-slots. Compound slots do not define tag space dimensions directly. Rather, they are are used to logically group other slots, and are defined using the `consists of` keyword.

```
Handling:
  consists of Storage, Transit, Authentication.
```

*4) Todo Slot:* A placeholder slot, containing no values. Slots of this type are useful for building tags spaces using a top-down approach. In the tag space below, the intellectual property aspects of a the policy will be defined later:

```
DataTags: consists of Handling, IntellectualProperty.
IntellectualProperty: TODO.
```

To improve user friendliness, slots and values can have an associated explanatory text. Unlike code comments, these texts are retained at runtime, and interview softwares can make them available to the interviewee. Explanatory text association is done by adding it, between square brackets, immediately after the name of the defined entity, like so:

```
Storage [The way data are stored on the server]:
 one of
  clear,
  encrypt [Encryption by the server, before storage],
  multiEncrypt
    [Encryption by both the client and the server].
```

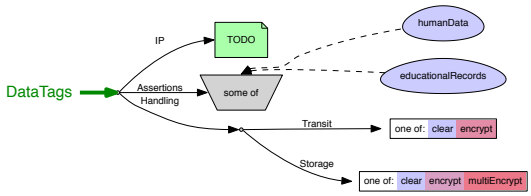


Fig. 3. The tag space defined in Listing 2, visualized by Tags' CliRunner tool. Atomic slots are visualized as scales, with the most lenient value on the left. Aggregate slots are shown as baskets, containing some of the aggregate values, visualized as eggs. Compound slots are visualized as circles with outgoing arrows. The arrows are labeled with the name of the slot they point to.

Slots are statically associated with the type implied by the slot definition. For example, the type of the `Storage` slot defined above contains the values `clear`, `encrypt` and `multiEncrypt`. It is an error to a assign a value to a slot, if said value is not a member of the slot's type.

The root slot of the tag space definition is a compound slot called `DataTags`. Unfolding a slot-based tag space definition to a list of dimensions is done as follows:

1) Start with a list containing the `DataTags` slot.
2) While there are compound slots in the list:
   a) Remove the first compound slot, $c$
   b) For each of $c$'s sub-slots (except the `TODO` ones), prefix the name the sub-slot with $c$'s name and a delimiter[3], and add the sub-slot to the list.
3) For each aggregate slot $a$ in the list:
   a) Remove $a$ from the list
   b) For each of $a$'s values, generate an atomic slot with the value's name prefixed by $a$'s name, and a delimiter. The generated slot has two values, the lesser representing the value not present in the aggregate slot, and the greater representing the value's presence. This pair of values can be thought of as `YES` and `NO`, or `applies` and `does-not-apply`, depending on the semantics of the values in aggregate slot.

When the algorithm terminates, the list of slots will contain only atomic slots with unique names, listing all dimensions of the described tag space, and the possible values for each dimension. The code addendum [5] offers a complete example for unfolding a tag space definition.

### B. Decision Graph

The "program" part of a Tags questionnaire is called a *decision graph*, and is modeled after a graph with various type of nodes. These types denote program instructions for asking questions, setting tag values, and control flow. During execution — *interview*, in Tags terms — the runtime engine traverses the decision graph, executing the command in each

[3]A delimiter is a character not allowed in a slot's name.

node. The runtime maintains two data structures: *Tags value*, a single instance of type `DataTags`, as defined in the tag space part of the questionnaire, and a call stack. Thus, a Tags interview can be viewed as an interactive, stacked traversal of the decision graph. At the end of the interview, the engine's Tags value contains the resultant data handling policy, and auxiliary assertions, if any.

Nodes are written between square brackets, and consist of a head and a body, separated by a colon (see Listing 1). The node head contains the node's instruction (e.g. `ask`, `call`) and a possible id. The node body, if present, contains additional information required for executing the node's instruction. For example, the body of a `call` contains the id of the node to call.

Listing 1. Sample nodes, showing various nodes structures: with and without id (lines 1 and 2, respectively), and a node with no body (line 3).

```
[call: medicalCompliance]
[>anId< set: Storage=encrypted]
[end]
```

Tags support various node types, some of which are listed below.

*a) Set Node:* Assigns values to some of Tags Value's slots. Atomic slots are assigned to using `=`, while aggregate slots are added to using `+=`. More than a single value can be assigned at a time, by separating the assignments using semicolon. For brevity, unique suffixes of a slot name can be used, instead of the full path to it. This type of abbreviation is demonstrated by the below example in the assignment to the `HIPAA` aggregate slot.

```
[set: DataTags/Handling/Storage=clear;
      HIPAA+={expertDetermination, waiver}]
```

A `set` node can only make values stricter — in effect, it has the semantics of the $\oplus$ operator, defined in II-A. This prevents different sections of an interview from lifting each other's restrictions: Consider a dataset containing mundane health data and private educational data. The educational data section of the questionnaire will conclude that the dataset has to be encrypted at rest. However, as the dataset does not contain private health data, the health section of the same questionnaire might allow it to be stored in the clear. Clearly, the dataset has to be stored encrypted. By preventing `set` nodes from replacing strict values with lenient ones, the health section cannot interfere with the decision made by the educational records section.

*b) Ask Node:* Present a question to the user, and wait for an answer. An `ask` nodes define a set of possible answers. `ask` nodes have a recursive structure: each answer has its own decision sub-graph, executed by the engine if that answer is selected.

```
[ask:
  {text: Do the data contain personally
         identifying information of humans?}
  {terms:
    {personally identifying information:
         This means the name, address, fingerprints...}}
  {answers:
    {yes, living: [set: livingPersons=yes]
                   [call: privacySection ] }
    {yes, deceased: [call: deceasedSection ] }
    {no: [call: nonHuman] }}]
```

As legal and technological questions can be daunting, it is possible to clarify terms that appear in the question. This is done by using the optional `terms` sub node.

Our experience is that a yes/no question, where one answer leads to a sub-series of questions, and the opposite just moves to the next question, is a common pattern in questionnaires. For example, a questionnaire may ask the user whether a dataset contains health-related data, calling the health records section only if she answers "yes". In effect, this translates to an `ask` node that has a sub-graph for one answer only. In order to support this, if a node contains only a single `yes` or `no` answer, the other answer is assumed to be implicit — it is added automatically, and points to the node that is syntactically after said `ask` node.

*c) Reject Node:* Terminates the interview, flagging the dataset as unacceptable. Supports cases where the dataset was obtained illegally, or when handling it would breach some law.

```
[reject: Possible FERPA violation due to
         failure to obtain written consent...]
```

*d) Todo Node:* A placeholder node, to be later replaced with an implementation. Useful for top-down implementation approaches, where the questionnaire structure is decided early on, but its parts are implemented gradually.

```
[>educationalRecords< todo: comply with FERPA]
```

*e) Call and End Nodes:* The equivalent of a procedure call in other programming languages, this node makes the engine traverse another part of the decision graph before moving on to the syntactically next node. When visited by the engine, a `call` node is pushed onto the call stack, and the engine begins a new traversal of the decision graph, starting from the node whose id is in the `call` node body. When the engine visits an `end` node, it pops the `call` node and moved to the node syntactically after it. If the engine visits an `end` node while its call stack is empty, the interview terminates.

```
[call: healthSection]
...
[>healthSection< ask ...]
[end]
```

## IV. Example Questionnaire

This section shows a sample questionnaire covering jurisprudence pertaining to educational records. While inspired by real questionnaires and actual legislation, this questionnaire is built for instructional purposes, and is of very little legal value. For a real life questionnaire covering HIPAA compliance, we refer the reader to the code addendum [5].

Listing 2 shows the tag space part of the questionnaire. The root compound slot, `DataTags`, has three sub-slots: `Handling`, covering a simple data handling policy, `Assertions`, storing information about the dataset that could be used later (e.g. for generating a data use agreement), and `IP`, which will cover intellectual property issues once implemented. A visualization of the tag space is presented at Figure 3.

Listing 3. Sample decision graph for handling educational records.

```
1 [ask:
2   {text: Do the data concern humans?}
3   {answers:
4     {yes: [set: Assertions+=humanData]
5            [ask:
6              {text: Does the data contain
7                     educational records?}
8              {answers:
9                {yes: [call: eduCompliance]}
10               {no:  [set: Transit=encrypt]}}]}}]
11 [set: Storage=clear; Transit=clear] <-- defaults
12 [todo: Handle IP issues here]
13 [end]
14 <* Educational Compliance Section *>
15 [>eduCompliance< ask:
16   {text: Was written consent obtained?}
17   {answers:
18     {no: [reject: Cannot handle educational records
19                   without written consent.]}
20     {yes: [set: Storage=encrypt; Transit=encrypt;
21                 Assertions+=educationalRecords]}}]
22 [end]
```

Listing 2. Tag space for the sample questionnaire.

```
DataTags: consists of Handling, Assertions, IP.
Handling: consists of Storage, Transit.
Assertions: some of humanData, educationalRecords.
Storage: one of clear, encrypt, multiEncrypt.
Transit: one of clear, encrypt.
IP[Intellectual Property]: TODO.
```

The interview decision graph is shown in Listing 3 and visualized in Figure 4. It has two components, the main one (lines 1–10) asks the user about the data subjects and, in case they are human, about the data type. Line 11 sets a default data handling policy — note that since `set` nodes cannot replace strict values with lenient ones, if the `Storage` slot contains `encrypt` or `multiEncrypt`, the `set` node at line 11 has no effect. Line 11 ends with an end-of-line comment: all text starting with `<--` is ignored up to the end of the line.

The second component of the decision graph, starting at line 15, ensures the dataset was collected after the data subjects have signed a consent form. If that was not the case, the dataset is rejected (lines 18–19; again, the actual law is more complicated than this). Line 14, above the consent section, has a block comment documenting that section. Block comments, surrounded by `<* *>`, can span multiple lines.

## V. Tools and Methodologies

Tags is a new language, but already has some tools for creating questionnaires and performing interviews. This section will introduce some of these tools, and discuss budding methodologies for questionnaire development.

### A. Tags CliRunner

The main tool for developing questionnaires is CliRunner. This is a "swiss-army knife" command-line application. First and foremost, it allows running interviews. For debugging a questionnaire, CliRunner supports call stack and interview trace inspections. Code validations for finding unreachable nodes, unused tag values and invalid `call` nodes are also available. With the aid of Graphviz [10], CliRunner can create visualizations of tag spaces and decision graphs. Figures 3 and 4 were generated this way. CliRunner is built using an open architecture to support easy extension. Like the rest of the Tags code, it is open source.

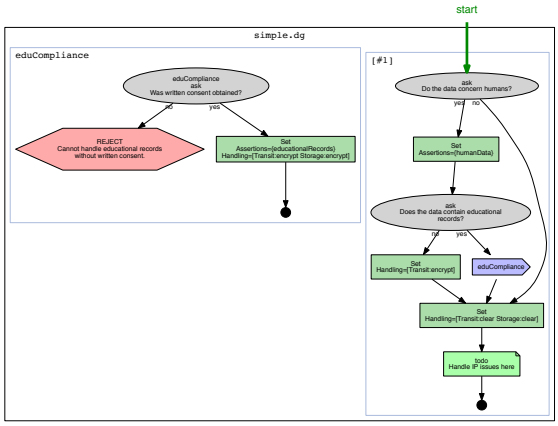


Fig. 4. The decision graph in Listing 3, visualized by Tags' CliRunner tool.

### B. Decision Graph Queries

Given a decision graph, it is possible to find all runs that yield policies with specific traits. For example, one can ask "which sets of answers result in a policy that allows clear transmission, and an assertion that the dataset holds sensitive data?". This type of question is useful, since the answer for most valid questionnaires should be "no such set of answers exists". CliRunner's `find-runs` command allows running such queries, by finding all the interviews (i.e. sequences of nodes and answers) that result in a tag value that is a superset of the supplied tag (tag value $a$ is a superset of tag value $b$ if they both agree on the values in $b$'s non-empty slots). Listing 4 shows the `find-runs` output for a query over the sample questionnaire presented in Section IV. We hope to expand this type of queries to a richer set of tools that will allow formal questionnaire verification.

Listing 4. CliRunner's `find-runs` command can find all the runs that result in a superset of a given result.

```
answer (? for help): \find-runs Storage=clear
Run 1:
[>[#1]< ask: Do the data concern humans?]
 -yes->
[>[#2]< set]
[>[#3]< ask: Does the data contain
                          educational records?]
 -no->
[>[#5]< set]
[>[#6]< set]
[>[#7]< todo: Handle IP issues here]
[>[#8]< end]
Final Tags:
DataTags/Assertions = humanData
DataTags/Handling/Storage = clear
DataTags/Handling/Transit = encrypt

Run 2:
```

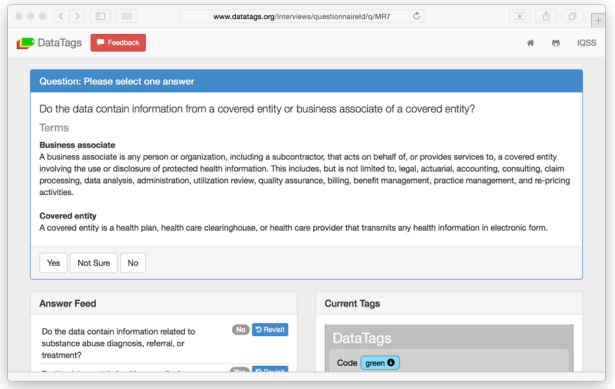


Fig. 5. A TaggingServer screen during an interview. Note the explanation of the terms, the answer feed (which allows users to revisit their answers), and the current tags, providing feedback on the tagging progress.

```
...

Found 2 matches in 3 possible runs.
```

### C. Web-based Interviews Using TaggingServer

TaggingServer is a web-based questionnaire runner, allowing users to go through a user-friendly interview online. Providing a good user experience for an interview is important, as answering a questionnaire which has legal and ethical implications might be a daunting experience.

TaggingServer provides application program interface (API), allowing it to be integrated into dataset depositing workflows of online data repositories, such as Dataverse [12] [7] [6]. The integration works as follows:

1) During the depositing process, the data repository requests an interview from a TaggingServer. The TaggingServer replies with a URL for the interview.
2) The repository redirects the user to the URL sent from TaggingServer.
3) The user is interviewed by the TaggingServer.
4) After the interview, the TaggingServer sends the result to the data repository. The data repository replies with a URL to redirect the user to.
5) The TaggingServer redirects the user to the provided URL. The data depositing process continues.

One important aspect of this integration is that the interview result is sent to the data repository directly from the TaggingServer, rather than manually by the user. This prevents users from fiddling with the interview result and from depositing one dataset under with the interview result of another. After the TaggingServer supplies the datatag to the repository, it is up to the repository to ensure that the dataset is handled according to the supplied tag, e.g. by using a sticky policy approach.

A TaggingServer hosting a questionnaire for creating data handling policies compliant with educational, health, and government records regulations, is available from http://datatags.org. Note that the tag space and questions were created as a proof of concept, demonstrating Tags' ability to handle large questionnaires that span multiple legislation areas. It is not intended for real world use.

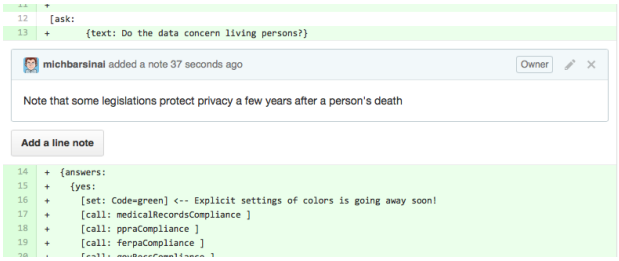


Fig. 6. Code review tools, such as GitHub's line comments shown here, allow experts to collaborate over a questionnaire.

### D. Methodologies

Tags questionnaires aim to provide legal and technical expertise. As such, developing them requires collaboration between legal scholars and security experts. Technical parts tend to have simple implications on the decision graph, mostly deciding between tag values such as encryption requirements and authentication method classes. Legal aspects, on the other hand, requires detailed question flows which may create complicated decision graphs.

Our questionnaire development process so far has been as follows: First, the relevant laws and regulations were studied, and an initial version of the tag space was constructed. Then, the decision graph was created, either using a word processor or a text editor. While the questionnaire was not written using Tags language, it did contain instructions with similar semantics, such as "set" or "go to question 12b". Once an initial version of the questionnaire was ready, it was translated to Tags, and visualized. The visualization was sent to the legal scholars for vetting. When errors were discovered, the legal scholars sent them back to the coding team. After a few iterations the questionnaire was loaded into a TaggingServer for final QA.

As Tags coding tools advances, we expect this methodology to change. CliRunner, and a language package for the Atom text editor, enable the development of questionnaires directly in Tags. Since Tags programs are plain texts, the development process can leverage distributed version control systems and code review tools for collaboration (see Figure 6).

Decision graph query tools, such as the `find-runs` command presented in Subsection V-B, enable an new set methodologies, inspired by formal software verification. For example, after studying the relevant jurisprudence and creating the tag space for the questionnaire, the development team can create a formal specification by composing a set of assertions that hold for any valid questionnaire. For example, such assertion might be "no series of answers exists such that the result of the questionnaire will contain the values `DataType=personal; Harm=criminal; Transit=clear`". This translates to `find-runs DataType=personal; Harm=criminal; Transit=clear` returning 0 results.

During the questionnaire development, the team can test it against the set of assertions. This can be seen as a form of test-driven design.

## VI. Related Work

Formalization of data handling policies is not new. W3C's recommendation Platform for Privacy Preferences Project (P3P) [1] allows web servers to state the data they collect and its intended use. It includes a policy matching concept, allowing users to specify which policies they agree to. Another aim of P3P is to allow users understand data handling policies. To this end, version 1.1 [19] (currently a W3C note) adds policy display guidelines. Unlike P3P, which focuses on web browsing scenario, ODRL [2] models policies used by commercial, educational, and government institutes. ODRL policy descriptions are composed of permissions and prohibitions of actions on assets, applying to a user with a defined role. To better support communities with specific needs, the ODRL core is extended by profiles. The PrimeLife Policy Language (PPL) [3], which focuses on downstream data usage, is another policy language that uses rules to permit and deny actions. Its *data handling preferences* (the way users expect their data to be handled), and *data handling policies* (the way servers treat the data) are comparable to Tags' $compliance()$ and $support()$ sub spaces. Data-Purpose Algebra [11] offers a more mathematical approach for modeling data usage restrictions. As ODRL, it aims to model general restrictions, created by legislation, contracts, or "common decency". DPA describes how data usage restrictions transform when one data item is produced by processing other data items.

The above projects, as well as Tags, aim to prevent forbidden usage. An alternative approach, presented in [17] and [18], is to hold users accountable for such usages.

Seneviratne, Kagal, and Berners-Lee demonstrated a tool for helping users comply with licenses when writing attributions for web content [14]. The design of TaggingServer, which offers a user friendly experience uncommon in data handling policy related tools, was in part inspired by commercial systems like TurboTax, which guide users through filling their tax forms.

## VII. Future Work

DataTags and legal modeling using DHP spaces are very young concepts, and we believe they can be developed in many ways. We are still lacking structured methodologies for developing tag spaces to model legislation and best practices. A study about the limits of this approach is also needed — what are the legal and technological concepts that cannot be captured by a DHP space?

While code tools go a long way in providing a collaborative environment, they still assume a certain technical knowledge, and are not as user friendly as word processors. We envision an on-line collaborative questionnaire development environment, allowing scholars from different backgrounds to develop, discuss, and publish questionnaires, no technical knowledge required. Additional tools can be developed, such as advanced graph queries, automated questionnaire verification, or automated detection of questionable data handling policies.

Finally, Tags' formal modeling may be useful as a tool for purposes other than dataset handling, such as comparative legal studies.

## Acknowledgment

The authors thank Sean Hooley, Alexandra Wood, David O'Brien, Stephen Chong, Salil Vadhan, Gary King, and the other members of the Privacy Tools Project at Harvard. We also thank the students of 2016 Fall semester at Ben-Gurion University of the Negev, Israel, who participated in class 202–1–4531 Topics in DataTags. This work was funded by grant CNS-1237235 from the National Science Foundation.

## References

[1] The platform for privacy preferences 1.0 (P3P1.0) specification. W3C recommendation, W3C, April 2002. http://www.w3.org/TR/2002/REC-P3P-20020416/.

[2] ODRL version 2.1 core model. http://www.w3.org/community/odrl/model/2.1/, 2015.

[3] Claudio A. Ardagna, Laurent Bussard, Sabrina De Capitani Di, Gregory Neven, Stefano Paraboschi, Eros Pedrini, Stefan Preiss, Dave Raggett, Pierangela Samarati, Slim Trabelsi, and Mario Verdicchio. Primelife policy language, 2009.

[4] Michael Bar-Sinai. DataTags code repository. https://github.com/IQSS/DataTaggingLibrary, 2016. Accessed: 2016-02-02.

[5] Michael Bar-Sinai, Latanya Sweeney, and Mercè Crosas. Technical appendix for datatags, data handling policy spaces and the tags language. http://dx.doi.org/10.7910/DVN/HKHGWZ, 2016.

[6] Mercè Crosas. The dataverse network®: an open-source application for sharing, discovering and preserving data. *D-lib Magazine*, 17(2), 2011.

[7] Mercè Crosas. A data sharing story. *Journal of eScience Librarianship*, 1(7), 2013.

[8] Cynthia Dwork. Differential privacy. In *ICALP*. Springer, 2006.

[9] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Third Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006.

[10] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *SOFTWARE - PRACTICE AND EXPERIENCE*, 30(11):1203–1233, 2000.

[11] Chris Hanson, Tim Berners-Lee, Lalana Kagal, Gerald J. Sussman, and Daniel J Weitzner. Data-purpose algebra: Modeling data usage policies. In *8th IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 173–177, 2007.

[12] Gary King. An introduction to the dataverse network as an infrastructure for data sharing. *Sociological Methods and Research*, 36:173–199, 2007.

[13] Crosas Mercè, Honaker James, King Gary, and Sweeney Latanya. Automating open science for big data. *ANNALS of the American Academy of Political and Social Science*, 659(1):260–273, 2015.

[14] Oshani Seneviratne, Lalana Kagal, and Tim Berners-Lee. Policy-aware content reuse on the web. In *International Semantic Web Conference*, volume 5823 of *Lecture Notes in Computer Science*. Springer, 2009.

[15] Latanya Sweeney. Operationalizing american jurisprudence for data sharing. Technical report, 2013.

[16] Latanya Sweeney, Mercè Crosas, and Michael Bar-Sinai. Sharing sensitive data with confidence: The datatags system. http://techscience.org/a/2015101601/, 2015.

[17] Daniel J. Weitzner, Harold Abelson, Tim Berners-Lee, Joan Feigenbaum, James Hendler, and Gerald Jay Sussman. Information accountability. *Commun. ACM*, 51(6):82–87, June 2008.

[18] Daniel J. Weitzner, Harold Abelson, Tim Berners-Lee, Chris Hanson, James A. Hendler, Lalana Kagal, Deborah L. McGuinness, Gerald Jay Sussman, and K. Krasnow Waterman. Transparent accountable data mining: New strategies for privacy protection. In *AAAI Spring Symposium: Semantic Web Meets eGovernment*. AAAI, 2006.

[19] Rigo Wenning and Matthias Schunter. The platform for privacy preferences 1.1 (P3P1.1) specification. W3C note, W3C, November 2006. http://www.w3.org/TR/2006/NOTE-P3P11-20061113/.