Towards a computer-interpretable actionable formal model to encode data governance rules

Rui Zhao
School of Informatics
University of Edinburgh
Edinburgh, UK
s1623641@sms.ed.ac.uk

Malcolm Atkinson
School of Informatics
University of Edinburgh
Edinburgh, UK
Malcolm.Atkinson@ed.ac.uk

***Abstract*—With the needs of science and business, data sharing and re-use has become an intensive activity for various areas. In many cases, governance imposes rules concerning data use, but there is no existing computational technique to help data-users comply with such rules. We argue that intelligent systems can be used to improve the situation, by recording provenance records during processing, encoding the rules and performing reasoning. We present our initial work, designing formal models for *data rules* and *flow rules* and the reasoning system, as the first step towards helping data providers and data users sustain productive relationships.**

***Index Terms*—data governance, data-usage rules, policy modelling, rule formalisation, compliance reasoning, provenance**

I. Introduction

Data ethics and privacy are of rising importance, especially with the establishment of GDPR [1]. Similar issues also apply in research when data from various sources are used as inputs to analyses and simulations. Researchers are aware that there are governance rules applied to the data, but they can easily lose track of the rules when the number of sources becomes large. The large volume of rules brings problem from three aspects:

1) to fully read and understand the rules;
2) to consider the consequence of combining data and their associate rules;
3) to assign rules to output so that results can be used compliantly.

One response is to make data open and freely accessible (e.g. under the FAIR principle [2] and/or as Linked Open Data as suggested by Tim Berners-Lee [3]). This sounds nice but it still leaves rules, for example to properly acknowledge sources and to protect personal and commercially sensitive data, even within collaborating communities [4]. Moreover, this doesn't solve (or even decrease) the prevalent *polarization*: data are either completely public (with one or a few well-known commonly agreed governance rules) or completely under control with heterogeneous (yet potentially similar) governance rules written in different languages, similar to the situation for copyright licenses.

This work has been accepted and should appear in the Proceedings of IEEE eScience 2019 Conference (BC2DC). Please cite the published work instead of this one when possible.

This issue becomes more serious when IoT devices (especially sensors) are widely used: data from them can be more sensitive, but users have limited control over where the data will go [5][6]. Therefore, it is necessary to let intelligent systems handle this as much as possible, while still getting people involved and ensuring they understand what is needed.

We propose to pioneer a combination of technology and its modes of use to help providers and users of data communicate precisely about the rules. We also set out to enable computer systems aid in compliance with rules while processing data. This would use automation to draw attention to rules at the relevant moments and collect information to support compliance with the rules. This requires that the notations are sufficiently machine readable and detailed to support the automation, and can be directly or indirectly (through automatic conversion) authored, understood and used by humans.

There are three approaches possible:

1) to police the system excessively taking little account of semantic details in order to prohibit every possible violation. This often excludes valid activities.
2) interpret the users' actions and inputs to verify in detail that rules will be honored. This is beyond the state of the art when users have the full power of the system.
3) encode the rules taking the semantics into account, and annotate the processing where necessary to perform reasoning. This requires extra work for both the data providers and the process developers.

Our work focuses on the third category which requires solutions to three issues:

1) How to write the governance rules in a computer-interpretable language?
2) How to handle the compliance checking?
3) How to ensure that malevolent actors cannot circumvent the rules?

Our work deals with the first two aspects, while the third is future work as it involves security and protection (for which research like [7] may be applied). In addition, the fact that the data processing is done by different bodies makes the problem inherently *federated*, so the solution should also be federated. Therefore, the solution needs to be:

1) vendor- and technology-agnostic;

2) able to extend easily to different disciplines;
3) interoperable across institutional and even national boundaries; and
4) not rely on a single trust authority.

These requirements direct our research to associate rules with data, rather than use a central system to control data access. A method to describe the effect of the processing steps on rules is also needed, in order to decide which rules need to be propagated or revised for the outputs. Therefore the solution includes the following:

1) **encoding procedure** a procedure to transform natural language governance rules to a formal representation;
2) **rule language** a computer-interpretable, extensible and interoperable formal language to write the data-governance rules;
3) **rule association** a standard protocol to associate encoded governance rules with the data;
4) **data flow representation** a vendor- and technology-agnostic method to represent and record data flow;
5) **flow behavior** a mechanism to allow processes to change the propagated governance rules;
6) **reasoning system** a reasoning system capable of working with any data-flow topology;
7) **correctness verification** a mechanism (or several mechanisms) to ensure or verify that all the steps above are conducted correctly.

Specifically, targets 2, 5 and 6 are the main aspects that our work and standardized provenance is used as the data-flow representation.

The structure of this paper is as follows: section II introduces the related research; section III describes the rule language we propose and provides some examples; section IV describes the design and architecture of our system, and presents initial results; section V highlights the contribution of our work and introduces the future work. A conclusion is drawn in section VI.

II. Related research

Here we describe the related research, stating its relation with the different aspects of our targets, as well as the limitations of each approach. It is worth noting that much research uses the word "policy" instead of "rule", and the part of the rules they capture lies mainly in controlling access.

A (now deprecated) standard called Platform for Privacy Preferences (P3P) was established by W3C in 2002, aimed at allowing websites to specify their privacy policies and compare those against user's preference to grant or prevent access (from the user's agent, e.g. web browsers). However, P3P had a very limited vocabulary and was not widely implemented in web browsers. It was finally deprecated and no successive standards were established.

Building on some concepts from P3P, E-P3P (Platform for Enterprise Privacy Practices) [8], one of the earliest works we found. It attempted to address the expressivity issue in a "formal" way by describing the privacy policy by specifying permitted data users, purposes and operations, and specifying consequent obligations. However, the set of predicates was very limited and the work payed no attention to the federated context, especially the interoperability issue. Checking compliance with policies associated with data was not completed there. It was reported in [9]; often cited as the foundation of the *sticky policy*.

Sticky policy [9][10] provided a conceptual framework to associate policies with data and "ensure" the data handlers (people, institutions, etc) are aware of the policies, by encrypting the data first, sending the policies with the encrypted data and sending the decryption key after checking that they agreed to comply with the rules. However, the checking procedure is not automatic and relies on the so-called Trusted Authority (some agencies explicitly trusted by the data owner). That said, in principle, the encrypt-and-decrypt concept can be used as the foundation of the protocol for an automated checking mechanism on each occasion and at each site he data is used.

CamFlow [11] is the work most similar to ours, though there is still much difference. It uses (Decentralized) Information Flow Control (IFC) [12] to represent the rules in the form of tags (in two groups, *secrecy* and *integrity*), which is a very simplified notation: compatibility of tags of the data and the process are checked before processing. On the other hand, CamFlow captures the importance of associating rules with data and allowing processes to change the associated rules (by specifying the modification to the output tags for each process), though it did not show any work in handling multi-input-multi-output processes. Moreover, CamFlow provides a mechanism to handle the data flowing between machines, making it possible to be used in a decentralized context.

Meta-code [13] is similar to CamFlow, and this technology is used in a series of works ([14]–[16]). They also use the IFC concept to assign permission tags (semantically, roles) to data, and then specify the flow behaviour (by specifying the *policy file of output data*) on the governance rule side (instead of the process specification side, like in CamFlow). In addition, they have a special *meta-code* part which is basically a program (resulting in either pass or error) executed at any specific file processing action (e.g. `onAccess`). However, because this is an arbitrary program, the policy of the resulting data can only be known at runtime, making it hard to do static analysis.

Thoth [17] on the other hand, favors the rule specification purely from the data originator side, because of its context: local data usage (*search engine* is used as an example). They have a set of logical connectives and predicates to specify the read, update and declassify policies. They do not use role tags like CamFlow and Meta-code do, but they allow the use of cryptographic keys (e.g. `sKeyIs(x)`) or IP addresses (e.g. `sIpIs(x)`) to identify data users and allow a special loop structure to test the match (existence). A further work, SHAI ([18]), provides a method to do static analysis for part of the rule language of Thoth. Despite the expressiviness, the fact that everything is based on the rules assigned by the data originators either requires them to be "omniscient" or prohibits many data uses that should be valid in principal.

Legal modelling is a whole different field, where precisely modelling the document is the priority. [19] and [20] uses reified predicate logic with Input/Output logic to model the legal corpus into their logical representation (e.g. GDPR[1]). [1] and [21] both looked at adding additional information to provenance to check the compliance of execution processes (against GDPR). However, these all focused purely on the modelling of terms, and never considered that the data will change (and so the rules will change) during processing.

There are also research and standards aimed at providing languages to express policies, with no binding to any processing systems, such as MPEG-21[2], XACML[3] and ODRL[4]. However, every one of them aims at expressing the policy for "the (specific) data", paying very limited attention (if any) to the policy for "derived data". From our perspective, Open Digital Rights Language (ODRL) is the one most closely related to our work. It provides an extensible semantic way to describe permissions, prohibitions and obligations as well as more fine-grained constraints. But because of its design priorities, data are explicitly specified and no mechanism is provided to support policy change, which means it does not directly meet our requirements. On the other hand, since ODRL provides its vocabulary in which many concepts are defined almost the same as ours, we will reuse some of its definitions in our language in the future.

III. Rule Language

Having compared the related research and described the general context, the requirements of the governance rule language are set as follows:

1) be precise enough (i.e. have no ambiguity);
2) be able to talk about more than just access controls;
3) be interoperable across institutional (and jurisdictional) boundaries.

Another language is also presented to model the flow behaviors, due to the necessity of referring to the governance rules. We call this the *flow rule* (language), and call the modelled governance rules *data rule*. As the name shows, the *data rule* is bound to the data, while the *flow rule* describes the process, as shown in Fig. 1.

In the following part, we first introduce our differentiation of "obligation and obligation definition"; the motivation and design of the *data rule* and *flow rule* languages are described below; the last subsection presents encoding examples.

A. Obligation and obligation definition

An underlying concern across our work is the differentiation between "obligation" and "obligation definition". This is introduced here to avoid confusion.

[1]dapreco, LegalRuleML formulae of deontic rules in the DAPRECO project.: Dapreco/daprecokb https://github.com/dapreco

[2]ISO/IEC TR 21000-1:2004 https://www.iso.org/standard/40611.html

[3]eXtensible Access Control Markup Language (XACML) Version 3.0 https://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

[4]ODRL Information Model 2.2 https://www.w3.org/TR/odrl-model/#policy-agreement

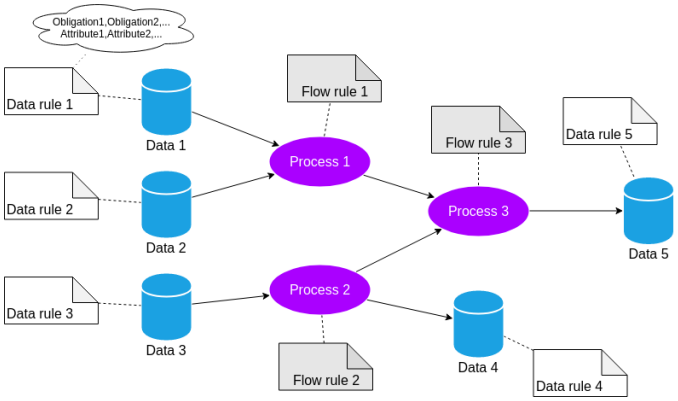


Figure 1: Annotation targets of *data rules* and *flow rules*

An "obligation" is an action that an agent (often a human) should perform. It can be bound to the agent, a piece of data, etc. It is an *action* that must be performed (otherwise it will be a violation).

On the other hand, an "obligation definition" specifies "what obligation will be activated under what situation". It is not an "obligation" itself, but describes how an obligation will be created.

Therefore, a key difference is that "obligation definitions" flow with data, while "obligations" may be "global" information. In fact, the current language only supports global[5] obligations.

To keep it short, this paper generally doesn't distinguish between these two terminologies when no confusion can be made. When necessary, we use "obligated action" to described "obligation".

B. Data rule

The data rule describes the governance rules of a dataset. It should capture the different aspects that a governance rule may talk about.

A large aspect missed in most of the reviewed related work is *obligation*. Examples include "*the third column of the dataset contains IP addresses, and should not be leaked to the outside*" and "*due to funding concerns, use of our data should be reported back to us*".

As shown in the examples above, the ability to describe "obligation" is important to model governance rules. Having this mechanism, we can mimic other types of rules (e.g. privacy control).

In our work, the concept of *obligation* is borrowed from [22]: the action that the data processor (normally human) needs to do. Specifically, the *post obligations* and *privacy policies* are the main focus of the current work.

In the original model, the obligation in [22] can contain an *identifier*, a *subject* (or a *role*) who the obligation is related to, an *action* which should be executed (i.e. the obligation), an *object* on which the obligation acts on, an *activation context* (condition) and a *violation context* (condition).

[5]"Global" in the **session** (see section III-B5).

Our model extends their model and makes adjustments to provide more freedom in writing the obligations and remove the unnecessary parts. In our work, the *identifier* need not be supplied (it is automatically generated), because the activated obligations will be stored as a list; the *object* is a part of the specification of the obligation; the *violation context* is not implemented at this stage. The *subject* of the obligation is considered as the data processor (assumed to be the person who started the session), so its specification only defines who the obligation may apply to (like a filter) and it can be merged into the *activation context*[6]. Therefore, the aspects that the obligation definition will need to talk about are:

- **obligated action** the action that the data processor needs to do when this obligation is activated;
- **validity binding** the data[7] dependency affecting the obligation definition;
- **activation condition** the condition that triggers the obligation.

The **obligated action** in our model contains both the *action* and *object* in the previous model, with more freedom to add additional information; the **validity binding** is a new concept in our model, specifically designed for the data-flow oriented point of view; the **activation condition** is almost the same concept as the *activation context*, with the addition of *subject* as a condition. In addition, to have a better syntax, **attribute** is introduced mainly to facilitate **validity binding** when used in flow rules (see III-C). They are all explained below.

1) Obligated action: To facilitate reuse and interoperability, we propose the **obligated action** to be an instantiation[8] of a *class* defined in an ontology. The arguments used to instantiate will also be specified in the data rule, using the **attribute** mechanism.

2) Validity binding: The **validity binding** is the data (or the part of the data) that this obligation applies to, and defaults to the whole dataset. If present, it refers to an **attribute**, and the meaning is: "this obligation definition is in force if and only if that specific attribute exists".

3) Activation condition: This element specifies the condition on which the obligation will be activated. Activated obligations will be handled separately (stored into a list, in the current implementation), but the obligation definition remains. Ideally, users should check the list of activated obligations after processing and conduct the activated obligations.

4) Attribute: **Attributes** are the extra information in the data rule, like *variables* in programming. They can be used (referenced) by the **obligated action** and the **validity binding**. An **attribute** contains a) an *identifier* (or id for short); b) a *value struct*. The identifier is an IRI (Internationalized Resource Identifier) and can be referenced (from **obligated action** and **validity binding**; the *value struct* is a struct (as in programming). In fact, there may be multiple **attributes** with the same *identifier*, so they will be merged into an ordered set of *value structs*, and be referenced with index.

5) Session: **Session** determines the checking scope of some **activation conditions**. For example, the `:OnImport` activation condition is `true` only when the rule appears in an *initial component* (see IV-B).

In the current implementation, **session** is handled very naively: one provenance graph is considered as one session. Normally, one execution of a workflow (in a workflow execution system supporting provenance) generates one provenance graph, so we can also say one workflow execution is one session. In practice, users run many workflows per sessions but sometimes re-attach a new session to a running workflow. This complexity is not relevant here.

C. Flow rule

Flow rules describe how the data rules propagate through a process in a data-flow graph. Therefore, the flow rules of all processes describe how the data rules of the data-flow graph inputs propagate to the outputs.

The flow rule of a process should reflect the important aspects of the actual data processing. For example, which outputs are related to which inputs, what data transformation has been done, etc.

The flow rule consists of two parts:

- *port mappings*;
- *refinements*.

The *port mappings* are the general behavior of how the data rules flow from inputs to outputs. They come in the form of input-output mappings (such as $(inputN, outputM)$), and should be interpreted as "every data rule from input N goes to output M (but may be refined)". The default mapping is "every data rule from every input port goes to every output port".

The (data) rule at the output may emerge modified to reflect processing. The modifications act on the **attributes**, and can be either *delete* or *edit*. The reason why *add* is not allowed at the moment is because **attributes** take effect only when referenced in obligations (so adding an dangling attribute makes no sense)[9].

The refinements need to specify the **attributes** they apply to. If the targeted **attribute** does not exist, this refinement simply passes without doing anything.

A *delete* refinement can either act on all **attributes** with a specified id, or a particular **attribute** with a specified *value struct*. On the other hand, an *edit* refinement can only act on an **attribute** with a specified id and a specified *value struct*, and it needs to specify a new *value struct* as the new value. Both *delete* and *edit* should specify which output port they

[6]The working context requires a user identification at the start of session. User who initiated action, such as running a specific workflow, is then the subject. This is recorded in the provenance and can be easily used.

[7]The "data" here is actually **attribute** (as described in the latter parts).

[8]The correct word should be "individual". But we use "instantiation" and "instance" to mean they comply with the OWL axioms defined in the class and only exist / make sense in the context of our system.

[9]Generally, an obligation (definition) is atomic but carries modifiable **attributes**. If the processing adds a significant new requirement that is shown by including a new obligation on an output. This is beyond the design of modifying **attributes**.

are acting on and (optionally) which input port the attribute is from.

With these mechanisms, it is possible to capture some useful information that would otherwise not be possible. For example, the 3rd column of input 2 is placed at the 5th column in output 1. Moreover, if a consecutive process removes column 5, we can now conclude the original 3rd column is removed in the output of the second process. This is especially useful if the original 3rd column contains sensitive information. Fig. 2 demonstrates this example.

After the refinement, *merging* is conducted. The very basic intuition for *merging* is: it is possible that rules from different input ports go to the same output port and these rules may contain the same **obligated action** class or the same **attribute** id (for various meaningful reasons, such as they were originally from the same dataset). Therefore, the objective of *merging* is to remove redundant rules and match **attribute** references.

Merging is essentially the merge of sets (the obligation set and the attribute set) between two (or more) resources. But because the obligation set contains references to the attribute set, it may need to be slightly altered. It executes as follows (see Algorithm 1 for a pseudo-code description):

1) for all attributes with the same id, remove redundancy and redirect the references (in the obligations) to the corresponding remaining attributes;
2) for all obligations, remove the obligations that are the same (i.e. having the same action, validity binding and activation condition)[10];
3) return the remaining attributes and obligations.

Algorithm 1 Merging algorithm (for each output port)

Require: rs is the list of rulesets (of data rules) to be merged
$o \leftarrow$ empty ruleset
for r in rs **do**
  add all in $r.obs$ to $o.obs$
  for $attr$ in $r.attrs$ **do** {attributes}
    if $attr$ in $o.attrs$ **then** {both the same id and value}
      $new\_attr\_ref \leftarrow$ reference to $attr$ in $o.attrs$
      for all ob in $o.obs$ such that ob references $attr$ **do**
        update reference of $attr$ to new_attr_ref
        if ob has duplicates **then**
          remove ob from $o.obs$
        end if
      end for
    else
      add $attr$ to $o.attrs$
    end if
  end for
end for
return o

[10]The *equality* is in the ontology sense: the whole IRI (including the namespace) should match. A more extended solution should take ontological reasoning into account (especially the `owl:sameAs` axiom), because two different bodies may use different ontologies and parameters to express the same concept.

D. Encoding examples

In this section, we demonstrate the encoding examples of some governance rules as obligations, including the two example rules given at the beginning of III-B.

For simplicity, we omit the ontology prefixes (namespaces). In fact, they are not used for special purposes (except for identifier) for this paper[11].

1) Keep third column secret: For the rule "*the third column of the dataset contains IP addresses, and should not be leaked to the outside*", the key information is that the specific column should be kept secret, so this can be encoded as:

```
Obligation(:secret :col3, :col3, )
Attribute(:col3, :column 3)
```

Here, `:secret` and `:column` are two ontology classes. We omit the prefixes (namespaces) but we keep the `:` to indicate this should be an ontological reference. The `:col3` is the id of an individual as a result of the `Attribute()` statement[12].

The `Attribute()` statement defines an attribute, with id `:col3`, and value `:column 3`. This id is the individual IRI, so it can be referenced in the `Obligation()` statement[13].

The `Obligation()` statement contains three elements, separated by comma. The first element is the **obligated action** and the form `:secret :col3` means an instance of the ontological class `:secret`, and the argument is the (dereferenced) value of (the attribute) `:col3` (i.e. `:column 3` initially). The second element is the **validity binding**, bound to the (same) `:col3` **attribute**. The third element is the **activation condition**, but is empty for this obligation.

An empty **activation condition** means this obligation will never be activated. As a consequence, we use this as a special case meaning the related information should be checked after processing (e.g. check all output data to see if any of them have the `:secret` obligation). A better syntax may be used as a future work.

2) Report data use: The stem of the rule "*due to funding concerns, use of the data should be reported back to us*" is actually "*use of the data should be reported back to us*", so this can be encoded as:

```
Obligation(:report :source, :source, :OnImport)
Attribute(:source, "Fictional source")
```

The `:OnImport` is the **activation condition** of this obligation, meaning this obligation will be activated when the data is first introduced to the execution session.

Specifically, our rule language enables the data provider to be explicit about what they mean by "use". The example above

[11]In the experiment, we developed a very primitive ontology. The ontology to use in a production system is still under development to provide better interoperability. The ontology to describe internal structures of data is external to our research.

[12]From the computational point of view, it is automatically created in the system.

[13]In fact, this attribute id doesn't have to be in the ontology world because it is not persistent. However, to make the syntax explicitly show this is not a *value*, we keep the `:` here and describe it as an individual for simplicity.

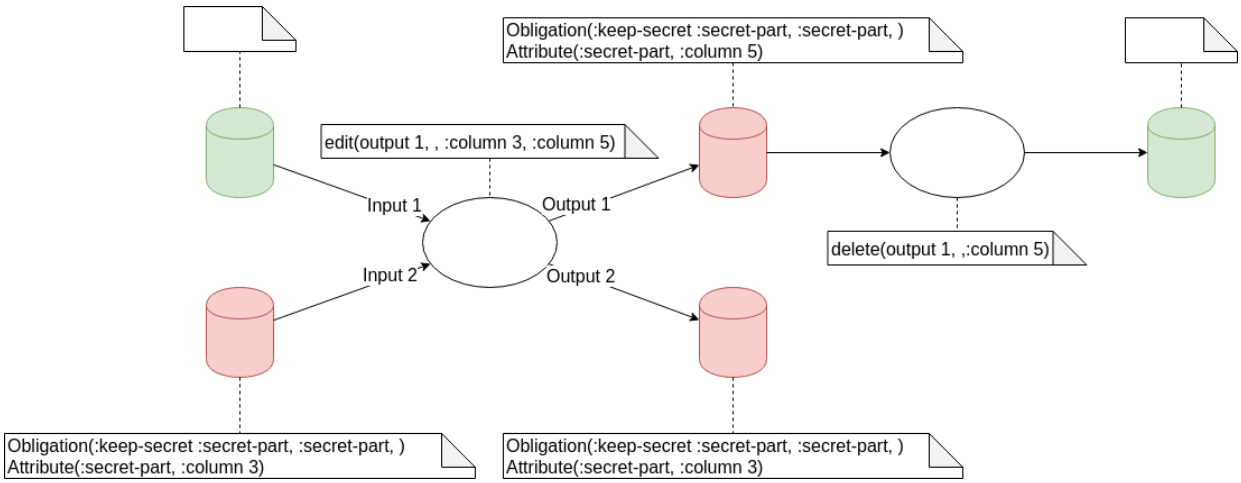


Figure 2: An example demonstrating the use of flow rules: here they show adaption of the data rule with the column number change. Ellipse for processes; cylinder for data; note-shape for rules; green for data without sensitivity; red for data with sensitivity; the flow rules here follow the default propagation (i.e. everything in to everything out).

defines "use" as "the first time as input to the workflow". An alternative definition could be "every time it is sent as an input to a component":

```
Obligation(:report :source, :source, :OnAsInput)
Attribute(:source, "Fictional source")
```

3) Acknowledge when publish: A common rule used in research is "*if you use our dataset and make a publication based on it, you should acknowledge this in your publication using a proper way such as 'XX'*". This could be encoded as:

```
Obligation(:acknowledge :form, , :OnPublish)
Attribute(:form, "XX")
```

The `:OnPublish` condition matches a component which is identified as making a publication. This component is actually a "virtual" component, meaning it is not a component in the provenance (of the executed workflow), but an additional element added to the provenance at runtime to assist the reasoning.

IV. System

Facing the research questions and taking lessons from existing research, our system provides a mechanism to model more than the access control aspect of the data governance rules in a federated context, using the rule language described in section III. In this section, we describe the system architecture (Fig. 3) and the design concerns.

A. Architecture

Since the system intends to provide a vendor- and technology-agnostic method to deal with rules, the foundation must have the same neutrality. The foundation contains two parts: a) data-flow representation; and b) rule encoding language. The rule languages are already described in section III. Hence, this section mainly describes the data-flow representation.

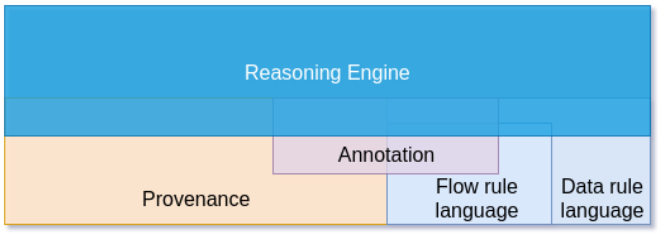


Figure 3: System architecture

The data-flow representation is the main input to our system (plus the encoded rules). Therefore, we choose *provenance* W3C PROV [23] to be the representation, taking the advantage that PROV-O builds on OWL [24] which allows the interoperability between different institutions by technologies like ontology matching. In fact, as mentioned above, our rule language also uses ontology for interoperability.

Specifically, we use S-Prov [25], an ontology extending W3C PROV-O to record data-streaming workflow executions, as the provenance specification. The reason we choose this ontology is because a) it provides necessary information to identify workflow components; b) our architecture is data–centric which aligns with the data-streaming perspective of S-Prov. However, the data-streaming feature is not used in our system for the moment, so in principle our system can also take other ontologies (e.g. OPMW[14]) as the input, either by directly having a separated parser or using ontology matching from the other representation to S-Prov.

Thus, our system uses the provenance to reconstruct the data flow and identify relevant information (e.g. the agent

[14]OPMW-PROV: The Open Provenance Model for Workflows http://www.opmw.org/index.html

who initiated the execution), as the information necessary for reasoning (e.g. in the **activation condition** of data rules).

Because our usage context is to help researchers comply with the rules, there is no need to "enforce" the rules in the system. Instead, a method such as a prompt is considered enough for the user (i.e. researcher) either as a reminder or a checklist. For example, a user can ask what obligations he has pending.

Therefore, the reasoning engine takes both the information from provenance and the rules as input to reason about:

1) the governance rules of the output data at each stage; and
2) the activated obligations.

These two missions are essentially the reason why we design these two sets of rules – they can be expressed as:

1) conduct the flow rules;
2) check the activation conditions of data rules (and instantiate the activated obligations).

B. Implementation details

Because the rule languages are not yet expressed in logic, we are unable to use formal reasoners (theorem provers, for example). Instead, we developed a computational reasoning system as proof-of-concept to demonstrate the feasibility of this reasoning since the semantics of the rule languages are clear.

The system takes the provenance as the input and then extracts the data flow to construct a directed graph – vertices are the processing steps (components) and the edges are the data transmission between components. SPARQL is used for this procedure, and the extra information like the `component function` is kept in the resulting graph. The flow rules are also inserted into the resulting graph.

After obtaining the graph, the reasoning can be conducted for every node with 0 in-degree and then repeatedly for the rest of the graph with 0 in-degree. This is essentially the same as doing a topological sorting based on the in-degrees and then performing reasoning from the beginning to the end.

For every component, the reasoning does the following:

1) Receive the data rules for every input port;
2) Identify the current context of execution and check the activation conditions of every data rule; If any activation condition is met, instantiate and store the activated obligation;
3) Propagate the data rules from the inputs to outputs in accordance with the flow rules.

In our implementation, the data rules are stored as an augmentation to the graph extracted from provenance, directly attached to the output ports. The reason why they are not attached to data is because S-Prov schema aims at the data-streaming style processing, so data are split into small chunks and therefore data is produced and transmitted from an output to an input incrementally. We assume that the governing rules are the same for every data unit in the stream.

Specifically, the data rules for the *initial components* (i.e. components that do not take any inputs) are *imported*. We use the so-called *recognizer* to identify what rules shall be *imported* by these components, and assign a fake input with these rules. In a real production system, this should be replaced by a more sophisticated way, but it is enough for the proof-of-concept.

Table I: Pattern coverage of each synthetic workflow

Graph	1-1	1-n	n-1	n-m
single line streaming	✓			
spreading	✓	✓		
collecting	✓		✓	
redispatching	✓		✓	✓

C. Initial results

We conducted a few experiments to verify the system's correctness and the model's feasibility.

We assume the data flow graphs are always fully connected Directed Acyclic Graphs (DAGs) with dangling input and output ports / edges (which are directed connected to input and output data). Therefore, the components are the vertices of the graph, the data-flow connections are the edges. The potential topology of DAGs is defined by its vertices and edges, and the type of vertices (in terms of edges) is finite, so we only need to ensure the correctness of our system for every one of these vertices (components):

1) one-input-one-output (1-1), to test the propagation;
2) one-input-multi-output (1-n), to test spreading;
3) multi-input-one-output (n-1), to test the aggregation;
4) multi-input-multi-output (n-m), to test redispatching.

In principle, in a DAG, a vertex can have zero inputs or zero outputs. Let aside the ones with dangling edges, semantically, a component with zero inputs is a producer: it does not rely on external resource, but will produce data (e.g. a prime number generator); a component with zero outputs makes no sense in a data flow (we argue a "storage" component also produces outputs) so we do not consider it at all. For experimental purposes, we use producers with one output port as the initial vertices, and connect the reset of the graph to them.
The synthetic workflows are written in dispel4py [26], a data-streaming workflow execution system. We executed these workflows with provenance generation (coupled with S-ProvFlow [25]), and then conducted reasoning. The reasoning intends to check that the propagation works correctly, so placeholders (simple strings) are used instead of meaningful data rules.

The vertex patterns checked by each synthetic workflow are shown in Table I. As a particular example, Fig. 4 demonstrates the structure of the original synthetic workflow of *redispatching*, and Fig. 5 shows the reasoning result.

The success of these experiments demonstrates the correctness of our system and model. It also demonstrates the feasibility of trying to handle data governance rules in any DAGs.

V. Contribution and future work

We describe the contribution of this paper and the future work in this section.

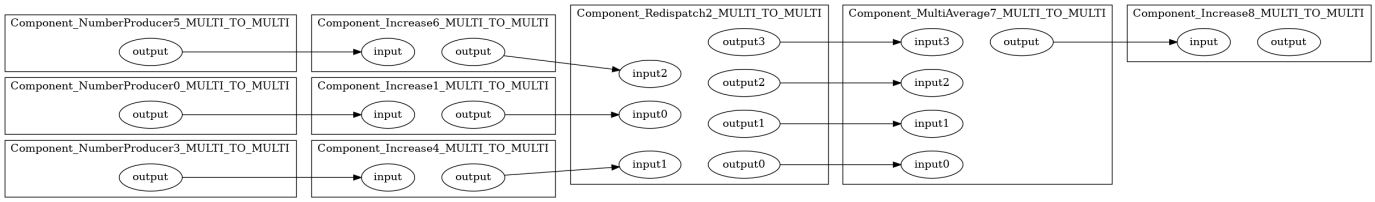
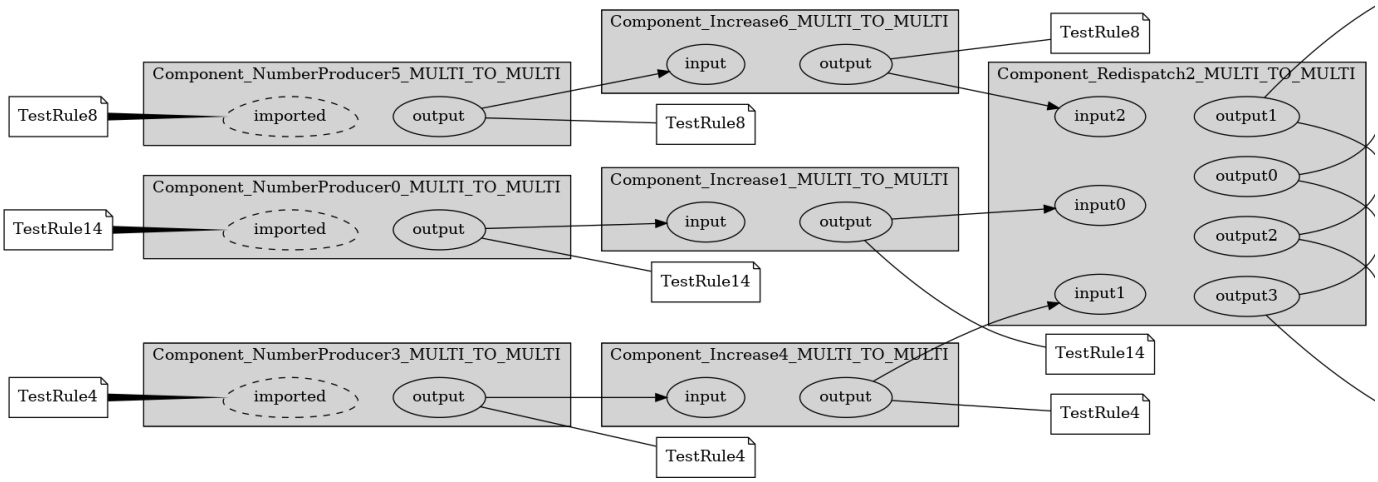
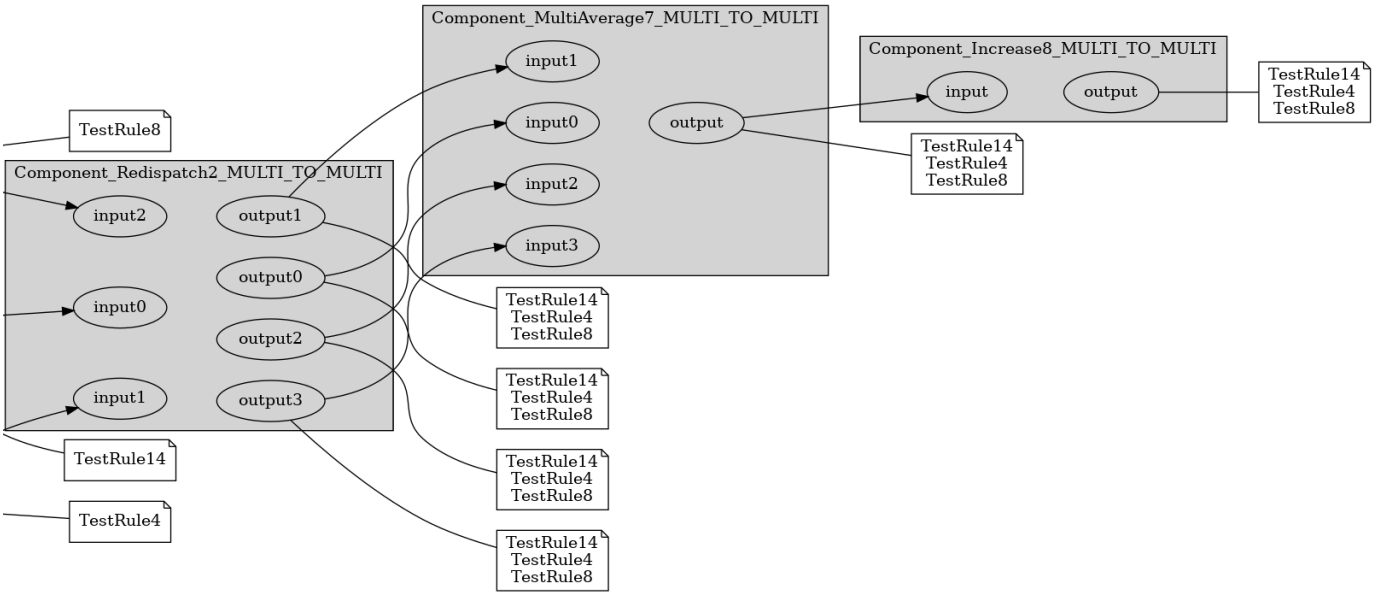


Figure 4: Topology of the "redispatching" synthetic workflow. Squares are components; ovals are ports; arrows are connections.



(a) left-half



(b) right-half

Figure 5: Reasoning result based on the provenance from executing the "redispatching" workflow (Fig. 4). Due to its width, the figure is split into two halves (sharing `Component_Redispatch2_MULTI_TO_MULTI`): subfigure (a) is the left-half; (b) is the right-half. In addition to Fig. 4: note-shapes are rules; lines (without arrows) connect the ports and the corresponding rules; black triangle arrows indicate *imported rules*; egg-shapes indicate virtual port; components are greyed.

A. Contribution

The main contribution of this paper lies in five aspects:

- clarified the questions concerning automating data-governance rules handling and identified the research targets in details in the *federated context*;
- demonstrated the use of provenance data as a source for rules;
- extended and refined the obligation model in [22] to better represent data-flow oriented data governance rule (policy) specification and the *federated context*;
- developed a method to describe the flow behavior of the rules for each process in a *multi-input-multi-output* data flow graph;
- pioneered a systematic method to help researchers comply with data governance rules in a federated research community.

B. Future work

Since we are addressing a field which has received little attention, there is a lot of research needed. This lies in three directions:

1) Ground the rule languages into logic;
2) Extend the language to capture more aspects;
3) Develop supporting technologies.

The following part describes these directions in more detail with our emphases. It's worth noting that these three directions interact. Specifically, explicit encoding of *requirements* (mentioned in section III) is related both to directions 2 and 3 (explained below): making the semantics explicit and providing a specific syntax in direction 2; providing the mechanism to check in direction 3.

1) Logic grounding: This direction will be our main focus in the next step, because a) we can become more certain that there is no internal problem with the rule language; b) existing formal reasoners can be used, taking the advantage of their proven correctness; c) the rules from different sources can be compared by checking their logical conflicts, enabling more use cases.

The data rules and flow rules shall be grounded into two logic sets, because of their different properties. For any formalisation, the data rules should always be harmonious with OWL because ontology (and therefore OWL) is critical to interoperability. Similarly, because our framework builds on top of semantic technology, the modelling mechanism provided by OWL axioms is also worth exploring to encode and validate flow rules.

2) Language extension: This direction is also important and will be our secondary target. More fine-granulated semantic meanings can be encoded with an extended language. This should better be conducted with direction 1.

One item of future work may be to extend the **activation condition** to accept more triggers; another potential item is to add more language semantics, such as capturing and denoting the *violation context*, or making *session* customisable. In the meantime, ODRL may be a candidate to encode some elements of data rules (e.g. **obligated action**).

3) Supporting technologies: This direction concerns the collaboration and system implementation scenario, and generally would be conducted with other researchers from application contexts. For example, our collaborator in KNMI will bring useful use cases for our framework supporting earth-system consortia.

An explicit direction may be to define a standard protocol to retrieve rules with data; real-time processing (instead of the current retrospective analysis) should also be investigated. This would enable alerts to be sent to users when new obligations emerge and eventually lead to warnings preventing serious rule infringement, exploiting active provenance [27].

VI. Conclusion

In this paper, we have identified a requirement for improving rule management to help users and providers work well together in a federated context. We proposed a method to help by delivering appropriate encoding and automation. This includes a formal model to encode the data governance rules, and a companion model to describe how the governance rules will flow and change during processing. We then presented some example encodings of governance rules using our model, and presented our proof-of-concept system as well as its initial results to demonstrate feasibility. Finally, we highlighted our contribution as a conceptual and practical framework.

We believe our work points to an important research direction, and provides a good first step towards solving that. More work will follow, as described in the future work section.

References

[1] B. E. Ujcich, A. Bates, and W. H. Sanders, "A Provenance Model for the European Union General Data Protection Regulation", in *IPAW*, 2018. DOI: 10.1007/978-3-319-98379-0_4.

[2] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, *et al.*, "The FAIR Guiding Principles for scientific data management and stewardship", *Scientific Data*, vol. 3, p. 160 018, Mar. 2016. DOI: 10.1038/sdata.2016.18.

[3] Tim Berners-Lee, *Linked Data - Design Issues*, Jun. 2009. [Online]. Available: https : / / www . w3 . org / DesignIssues/LinkedData.html (visited on 07/08/2019).

[4] Y. Demchenko, C. Blanchet, C. Loomis, *et al.*, "CYCLONE: A Platform for Data Intensive Scientific Applications in Heterogeneous Multi-cloud/Multi-provider Environment", in *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*, Apr. 2016, pp. 154–159. DOI: 10.1109/IC2EW.2016.46.

[5] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, "Security, privacy and trust in Internet of Things: The road ahead", *Computer Networks*, vol. 76, pp. 146–164, Jan. 2015. DOI: 10.1016/j.comnet.2014.11.008.

[6] S. M. R. Islam, D. Kwak, M. H. Kabir, M. Hossain, and K. Kwak, "The Internet of Things for Health Care: A Comprehensive Survey", *IEEE Access*, vol. 3, pp. 678–708, 2015. DOI: 10.1109/ACCESS.2015.2437951.

[7] P. Missier, S. Bajoudah, A. Capossele, A. Gaglione, and M. Nati, "Mind My Value: A Decentralized Infrastructure for Fair and Trusted IoT Data Trading", in *Proceedings of the Seventh International Conference on the Internet of Things*, ser. IoT '17, New York, NY, USA: ACM, 2017, 15:1–15:8. DOI: 10.1145/3131542.3131564.

[8] G. Karjoth, M. Schunter, and M. Waidner, "Platform for Enterprise Privacy Practices: Privacy-Enabled Management of Customer Data", en, in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, Apr. 2002, pp. 69–84. DOI: 10.1007/3-540-36467-6_6.

[9] M. C. Mont, S. Pearson, and P. Bramhall, "Towards accountable management of identity and privacy: Sticky policies and enforceable tracing services", in *14th International Workshop on Database and Expert Systems Applications, 2003. Proceedings.*, Sep. 2003, pp. 377–382. DOI: 10.1109/DEXA.2003.1232051.

[10] S. Pearson and M. Casassa-Mont, "Sticky Policies: An Approach for Managing Privacy across Multiple Parties", *Computer*, vol. 44, no. 9, pp. 60–68, Sep. 2011. DOI: 10.1109/MC.2011.225.

[11] T. F. J.-M. Pasquier, J. Singh, D. Eyers, and J. Bacon, "CamFlow: Managed Data-sharing for Cloud Services", *IEEE Transactions on Cloud Computing*, vol. 5, no. 3, pp. 472–484, Jul. 2017, arXiv: 1506.04391. DOI: 10.1109/TCC.2015.2489211.

[12] A. C. Myers, A. C. Myers, and B. Liskov, "A Decentralized Model for Information Flow Control", in *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '97, event-place: Saint Malo, France, New York, NY, USA: ACM, 1997, pp. 129–142. DOI: 10.1145/268998.266669.

[13] Håvard D. Johansen, Eleanor Birrell, Robbert van Renesse, *et al.*, "Enforcing Privacy Policies with Meta-Code", en, ACM Press, 2015, pp. 1–7. DOI: 10.1145/2797022.2797040.

[14] H. D. Johansen, W. Zhang, J. Hurley, and D. Johansen, "Management of body-sensor data in sports analytic with operative consent", in *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, Apr. 2014, pp. 1–6. DOI: 10.1109/ISSNIP.2014.6827638.

[15] J. Hurley and D. Johansen, "Self-Managing Data in the Clouds", in *2014 IEEE International Conference on Cloud Engineering*, Mar. 2014, pp. 417–423. DOI: 10.1109/IC2E.2014.31.

[16] A. T. Gjerdrum, H. D. Johansen, and D. Johansen, "Implementing Informed Consent as Information-Flow Policies for Secure Analytics on eHealth Data: Principles and Practices", in *2016 IEEE First International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, Jun. 2016, pp. 107–112. DOI: 10.1109/CHASE.2016.39.

[17] E. Elnikety, A. Mehta, A. Vahldiek-Oberwagner, D. Garg, and P. Druschel, "Thoth: Comprehensive Policy Compliance in Data Retrieval Systems", en, in *Proceedings of the 25th USENIX Conference on Security Symposium*, ser. SEC'16, Berkeley, CA, USA: USENIX Association, Aug. 2016, pp. 637–654.

[18] Eslam Elnikety, Deepak Garg, and Peter Druschel, "Shai: Enforcing Data-Specific Policies with Near-Zero Runtime Overhead", *arXiv:1801.04565 [cs]*, Jan. 2018, arXiv: 1801.04565.

[19] L. Robaldo, L. Humphreys, X. Sun, *et al.*, "Combining Input/Output logic and Reification for representing real-world obligations", en, *Proceedings of the 9th International Workshop on Juris-informatic (JURISIN 2015)*, 2015.

[20] L. Robaldo and X. Sun, "Reified Input/Output logic: Combining Input/Output logic and Reification to represent norms coming from existing legislation", en, *Journal of Logic and Computation*, vol. 27, no. 8, pp. 2471–2503, Dec. 2017. DOI: 10.1093/logcom/exx009.

[21] H. J. Pandit, D. O'Sullivan, and D. Lewis, "Queryable Provenance Metadata For GDPR Compliance", en, *Procedia Computer Science*, p. 7, 2018.

[22] Y. Elrakaiby, F. Cuppens, and N. Cuppens-Boulahia, "Formal enforcement and management of obligation policies", *Data & Knowledge Engineering*, vol. 71, no. 1, pp. 127–147, Jan. 2012. DOI: 10.1016/j.datak.2011.09.001.

[23] P. Groth and L. Moreau, *PROV-overview. An overview of the PROV family of documents*, 2013. [Online]. Available: https://www.w3.org/TR/prov-overview/.

[24] D. L. McGuinness, F. Van Harmelen, *et al.*, *OWL web ontology language overview*, 2004. [Online]. Available: https://www.w3.org/TR/owl-features/.

[25] A. Spinuso, "Active provenance for data intensive research", en, PhD Thesis, University of Edinburgh, Nov. 2018.

[26] R. Filguiera, I. Klampanos, A. Krause, *et al.*, "Dispel4py: A Python Framework for Data-intensive Scientific Computing", in *Proceedings of the 2014 International Workshop on Data Intensive Scalable Computing Systems*, ser. DISCS '14, Piscataway, NJ, USA: IEEE Press, 2014, pp. 9–16. DOI: 10.1109/DISCS.2014.12.

[27] A. Spinuso, M. Atkinson, and F. Magnoni, "Active provenance for Data-Intensive workflows: engaging users and developers", in *Proc. eScience BC2DC workshop*, 2019.