SPECIAL

Scalable Policy-awarE Linked Data arChitecture for privacy, trAnsparency and compLiance

Deliverable D2.7

Transparency Framework V2

Document version: V1.0

SPECIAL DELIVERABLE

Name, title and organisation of the scientific representative of the project's coordinator:

Ms Jessica Michel t: +33 4 92 38 50 89 f: +33 4 92 38 78 22 e: jessica.michel@ercim.eu

GEIE ERCIM, 2004, route des Lucioles, Sophia Antipolis, 06410 Biot, France

Project website address: http://www.specialprivacy.eu/

Project	
Grant Agreement number	731601
Project acronym:	SPECIAL
Project title:	Scalable Policy-awarE Linked Data arChitecture for prIvacy, trAnsparency and compLiance
Funding Scheme:	Research & Innovation Action (RIA)
Date of latest version of DoW against which the assessment will be made:	17/10/2016
Document	
Period covered:	M1-M23
Deliverable number:	D2.7
Deliverable title	Transparency Framework V2
Contractual Date of Delivery:	30-11-2018
Actual Date of Delivery:	30-11-2018
Editor (s):	Sabrina Kirrane (WU)
Author (s):	Sabrina Kirrane (WU), Uroš Milošević (TF), Javier D. Fernández (WU), Axel Polleres (WU), Jonathan Langens (TF)
Reviewer (s):	Rigo Wenning (ERCIM), Piero Bonatti (CeRICT), Wouter Dullaert (TF)
Participant(s):	WU, TF, ERCIM, CeRICT
Work package no.:	2
Work package title:	Policy and Transparency Framework
Work package leader:	WU
Distribution:	PU
Version/Revision:	1.0
Draft/Final:	Final
Total number of pages (including cover):	54

Disclaimer

SPECIAL has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731601.

Contents

List of Figures

1 Summary

The objective of the SPECIAL transparency and compliance framework is to help data processors and controllers to understand how the components and know-how developed in SPECIAL can be leveraged within a typical Enterprise setting. Core components of the SPECIAL transparency and compliance framework include: (i) the schema and vocabularies that can be used to express usage policies and data processing and sharing events; (ii) and the corresponding usage control middleware that is needed to interact with such data; and (iii) periphery components required in order to hook into Enterprise Line of Business and Business Intelligence / Data Science systems.

This aim of this deliverable is to define the scope of the SPECIAL transparency and compliance framework, which is used to guide the implementation of the SPECIAL platform and components, and also serves as a reference point for the open research challenges that we address in SPECIAL and how they relate to one another.

This deliverable builds upon the SPECIAL policy language which is described in *D2.1: Policy Language V1*. While, related information on the compliance checking, distributed ledger technology, and big data processing can be found *Deliverable D2.8 Transparency and Compliance Algorithms V2*.

In *Chapter* 1 we provide a high level overview of the SPECIAL landscape. After setting the scene in terms of the data sources, middleware and applications, we discuss the key role of Personally Identifiable Information (PII) in terms of Data Governance and identify open challenges that we aim to address in SPECIAL in *Chapter* 2. Finally, in *Chapter* 3, we provide our initial proposal for how the Resource Description Framework can be used to represent data processing and sharing events, by describing the key terms and their relationship both to one another and to the terms specified in SPECIAL usage policies.

Chapter 1

Policy, Transparency and Compliance Framework

In *D1.7 Policy, transparency and compliance guidelines V2* we identified several considerations and open questions with respect to the intersection between existing company systems and SPECIAL components. In this section, we frame the SPECIAL policy, transparency and compliance components developed in SPECIAL within the wider scope of a general Enterprise setting.

Enterprises rely on operational systems, commonly known as Line Of Business (LOB) applications, to perform day-to-day activities efficiently. For example, interactions with clients are recorded in Customer Relationship Management (CRM) applications, employee information is maintained in Human Resources (HR) applications and project documentation is stored in a Document Management Systems (DMS). When it comes to strategic decision making the data from LOB applications are ususally integrated and stored in a data warehouse that can be used for Business Intelligence (BI) / Data Science (DS) across the organisation.

1 SPECIAL Landscape

The purpose of the proposed policy, transparency and compliance framework depicted in *Figure* 1.1, and discussed below, is threefold: (i) to better understand the intersection between SPECIAL and existing company systems; (ii) to define the scope of the SPECIAL transparency and compliance work; and (iii) to serve as a framework that can be used to compare and contrast alternative solutions.

Components that are coloured in green are assumed to exist already, while components in blue will be developed by SPECIAL and/or the know how to develop said components will be provided by SPECIAL. In addition, the RDF symbol is used to denote RDF data, HDFS and Spark are used to highlight big data and big data processing respectively, and a simple reasoning symbol is used to denote components could potentially require some form of reasoning capabilities.

1.1 Data sources

The framework contains four different data sources. Two of which we assume already exist as they are necessary to support business operations (i.e. Line of Business Data), and strategic decision making (i.e. Business Intelligence / Data Science Data). In addition, we propose two

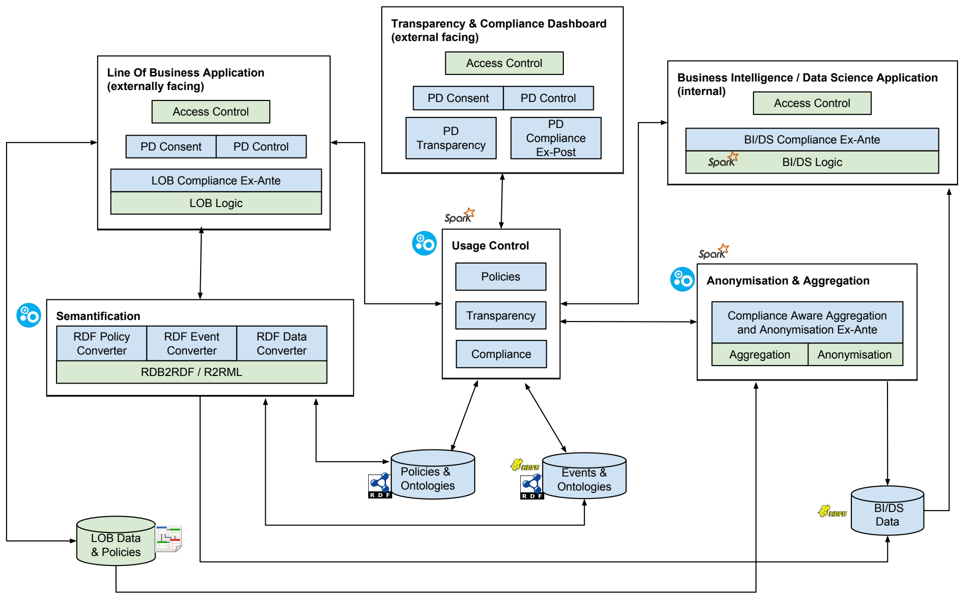

Figure 1.1: SPECIAL Landscape

additional data sources one which is used to store the consent, regulatory and business *policies* and another to store the data processing or sharing *events*. Both policy and the event data in SPECIAL will be represented as RDF. In our framework we have chosen to logically represent the data in two separate data stores as it is assumed that we will need to deal with a high volume of event data, which is most likely not the case for the policy data. It's worth noting that policy information will also need to be attached to the data stored in the BI/DS Data repository.

For additional information on the policy language the reader is referred to *D2.1 Policy Language V1*. While, details the LOB and BI/DS Data can be found in *Chapter* 2 *Personal Data Inventory*, which describes the SPECIAL personal data inventory strategy. Finally, information on the event log can be found in *Chapter* 3 *Linked Provenance/Event Information* of this deliverable.

1.2 Middleware

Beside the *Usage Control* middleware which is a core component of the policy, transparency and compliance framework, we propose two additional middleware components, namely *Semantification*, and *Anonymisation & Aggregation*.

1.2.1 Usage control

The usage control middleware is responsible for managing access to policies, event data and the respective ontologies. In *D2.8 Transparency and Compliance Algorithms V2 (Chapter 2)* we explore a number of *distributed architectures* that could potentially be used to store the event data. In addition in *D2.8 Transparency and Compliance Algorithms V2 (Chapter 3)* we introduce the SANSA Stack and discuss how it can be used to both query event data and to verify compliance of data processing an sharing events, at scale.

Irrespective of how the event data is stored (e.g. in a local, global or distributed ledger), in order to ensure non-repudiation by any of the involved peers (i.e. those owning, disclosing, and acquiring data), it must be possible to ensure that all recorded transactions have actually taken place and the autogenerated provenance/event data is tamper proof. As such we also examine the guarantees offered by *fair exchange* protocols in terms of non-repudiation of data sharing events in *D2.8 Transparency and Compliance Algorithms V1 (Chapter 4)*.

1.2.2 Semantification

Considering the variety of different LOB and BI/DS systems and database schemas, there is a need for RDF Converters that are capable of translating policies, events and possibly also LOB data into RDF. Well know approaches include Ontology-based Data Access (OBDA) in general and RDB2RDF and R2RML in particular. Updates over OBDA or even updates to Linked Data under ontological entailment are under-researched research topics within the Semantic Web Community. One of the main challenges where is understanding what is personal data, what policies are attached to that data and how are data processing and sharing events currently recorded, so that such information can be mapped to RDF. Here we are working on techniques that can be used for data discovery and cataloging. Additional details on the semantification of LOB and BI/DS Data can be found in *Chapter* **??** *Personal Data Inventory*.

1.2.3 Anonymisation and aggregation

Assuming that some BI/DS systems may depend on anonymisation and aggregation techniques it would be useful to be able to cater for policy-aware data anonymisation and aggregriation. The research in this area includes using machine learning techniques to verify existing anonymisation techniques, such as k-anonymity, l-diversity, t-closeness, to name but a few. This will improve the capability to avoid de-anonymisation and the ability to single out yet unknown persons for further discrimination. SPECIAL also aims to investigate how existing policies can be used to inform the anonymisation and aggregation algorithms.

1.3 Applications

In the proposed framework we have identified three different types of applications, namely, transparency and compliance dashboards, operational LOB applications, and strategic BI/DS applications.

1.3.1 Transparency and compliance dashboard

The transparency and compliance dashboard should be developed in such a way that it tackles the users' cognitive limitations. Key functions could include: obtaining consent in a non intrusive manner, presenting data processing and sharing events in a easily digestible manner, and enabling the user to manage existing consent for processing and sharing. *D6.3: Plan for community group and standardisation contribution* points to mashups as one potential means to support the integration of data coming from several different sources (most likely under the control of different controllers/processors). Although authentication and authorisation would highly depend on the existing company infrastructure, Web Identity and Discovery (WebID), which is a mechanism used to uniquely identify and authenticate a person, company, organisation or other entity, by means of a URI, could potentially be used for authentication across

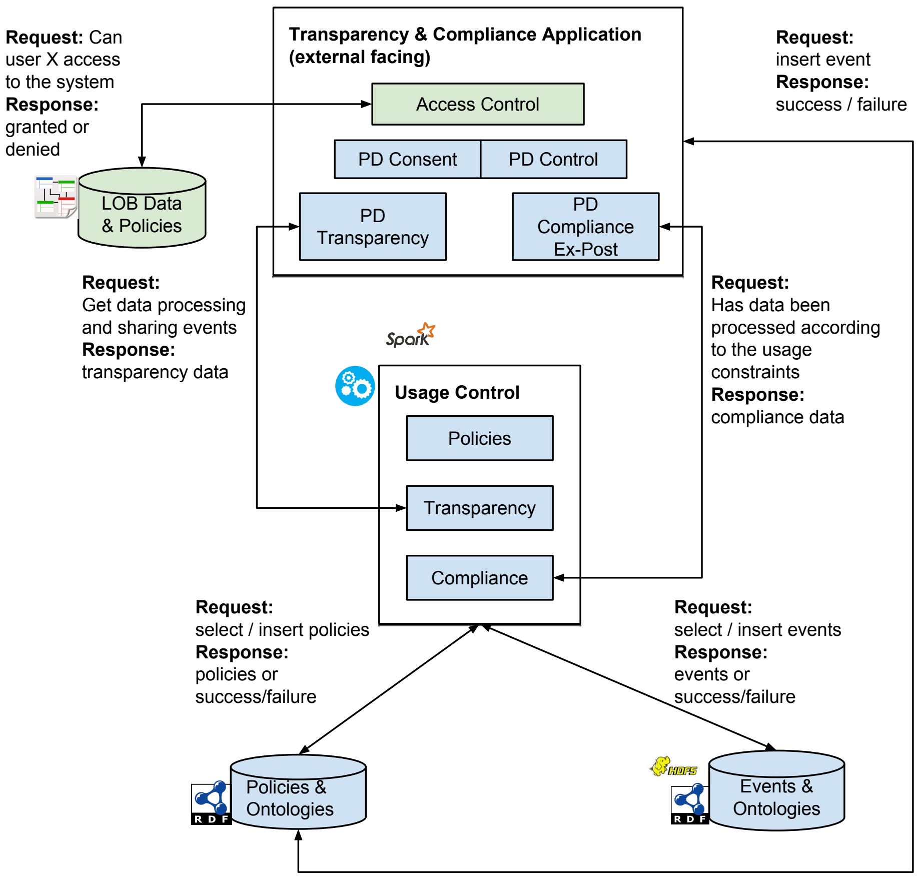
Request: Can user X access to the system
Response: granted or denied
LOB Data & Policies
Transparency & Compliance Application (external facing)
Access Control
PD Consent
PD Control
PD Transparency
PD Compliance Ex-Post
Request: insert event
Response: success / failure
Request: Get data processing and sharing events
Response: transparency data
Spark
Usage Control
Policies
Transparency
Compliance
Request: Has data been processed according to the usage constraints
Response: compliance data
Request: select / insert policies
Response: policies or success/failure
Request: select / insert events
Response: events or success/failure
Policies & Ontologies
Events & Ontologies
RDF
HDFS

Figure 1.2: Transparency and compliance application

different enterprises. Highlevel details of the interaction between the Transparency & Compliance Application, the Usage Control middleware and the Policies and Events data stores are presented in *Figure* 1.2.

1.3.2 Line of Business applications

Clearly there is a tight coupling between SPECIAL and existing Line of Business applications in terms of access control, consent and compliance checking. As mentioned earlier, authentication and authorisation would highly depend on the existing company infrastructure. Additionally, the data that will form part of the consent request and subsequently the usage policy needs to be based on the type of personal data required by the company in terms of product or service provision, and contextual information relating to the purpose, processing and sharing. Similarly, companies need to ensure that personal data processing and sharing within the orgaisation and by its Information Technology (IT) systems complies with relevant usage policies. Here there is a need to investigate and come up with a strategy for hooking into existing LOB applications. Additional details can be found in *Chapter* 2 *Personal Data Inventory*. Highlevel details of the interaction between the Line of Business Application, the Usage Control and Semantification

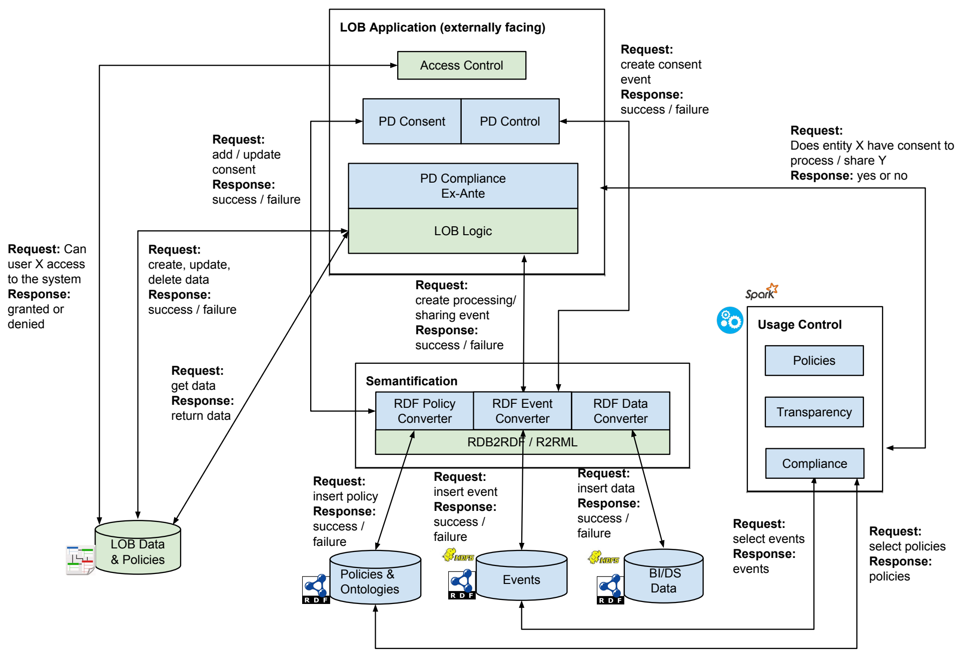
LOB Application (externally facing)
Access Control
PD Consent
PD Control
PD Compliance Ex-Ante
LOB Logic
Request: create consent event Response: success / failure
Request: Does entity X have consent to process / share Y Response: yes or no
Request: add / update consent Response: success / failure
Request: Can user X access to the system Response: granted or denied
Request: create, update, delete data Response: success / failure
Request: create processing/ sharing event Response: success / failure
Request: get data Response: return data
Semantification
RDF Policy Converter
RDF Event Converter
RDF Data Converter
RDB2RDF / R2RML
Usage Control
Policies
Transparency
Compliance
Request: insert policy Response: success / failure
Request: insert event Response: success / failure
Request: insert data Response: success / failure
Request: select events Response: events
Request: select policies Response: policies
LOB Data & Policies
Policies & Ontologies
Events
BI/DS Data

Figure 1.3: LOB application

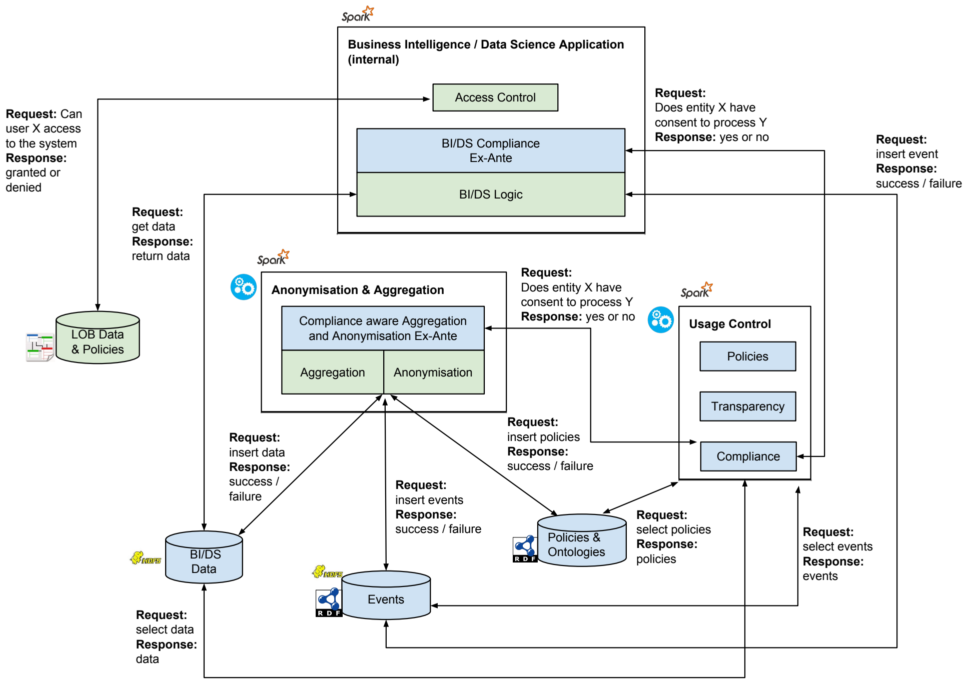
Business Intelligence / Data Science Application (internal)
Access Control
BI/DS Compliance Ex-Ante
BI/DS Logic
Request: Can user X access to the system Response: granted or denied
Request: Does entity X have consent to process Y Response: yes or no
Request: insert event Response: success / failure
Request: get data Response: return data
Anonymisation & Aggregation
Compliance aware Aggregation and Anonymisation Ex-Ante
Aggregation
Anonymisation
Request: Does entity X have consent to process Y Response: yes or no
Usage Control
Policies
Transparency
Compliance
LOB Data & Policies
Request: insert data Response: success / failure
Request: insert policies Response: success / failure
Request: insert events Response: success / failure
Request: select policies Response: policies
Request: select events Response: events
BI/DS Data
Policies & Ontologies
Events
Request: select data Response: data

Figure 1.4: BI/DS application

middleware, and the Policies and Events data stores are presented in *Figure* 1.3.

1.3.3 Business Intelligence / Data Science applications

As per the LOB applications there is a tight coupling between SPECIAL and existing Business Intelligence / Data Science applications, however here the focus is mainly on access control, and compliance checking. As before authentication and authorisation would highly depend on the existing company infrastructure. Additionally, companies need to ensure that personal business intelligence and data science within the orgaisation complies with relevant usage policies. Here again there is a need to investigate and come up with a strategy for hooking into existing BI/DS applications. In this deliverable, we first explore how usage control, transparency and compliance checking can be added to existing LOB applications. Highlevel details of the interaction between the Business Intelligence / Data Science Application, the Usage Control and Anonymisation & Aggregriation middleware, and the Policies and Events data stores are presented in *Figure* 1.4.

Chapter 2

Personal Data Inventory

Semantification of data is only possible under the assumption that the controller is aware of its existence, in all its forms, as well as its precise location. One way to ensure this is by cultivating the organisational behavior necessary to successfully manage data as an asset within the company itself. However, formal Data Governance implies establishing stakeholder agreement on data definitions, developing policies and procedures, encouraging data stewardship practices at multiple levels within the organisation, and continuous and active engagement in organisational change management processes. The time and effort required to accomplish that goal, along with the accompanying cost, are the prime reasons why true Data Governance remains out of reach for many enterprises.

Moreover, in large and fast changing/growing environments, the inability to maintain the required pace translates into inconsistent knowledge, which consequently has a negative impact on decision making and, ultimately, results in loss of trust in the solution. Hence, automation is desirable.

1 Data Discovery

Automated personal data discovery is certainly not a novel concept in enterprise data management. Nevertheless, traditional approaches to discovery are often plagued by both technical and legal limitations as they fail to acknowledge the requirements of the real-world applications, as well as the new legal framework. More specifically, built long before the GDPR era, such solutions still focus on *personally identifiable information* (PII), rather than the broader, GDPR, concept of *personal data*. Relying mostly on rules, regular expressions, and named entity databases, which lack both context and semantics, PII discovery tools will not work for:

- Every possible scenario, as there will always be exceptions (for example, foreign national numbers);
- Data that the controller has not considered or is simply unaware of (for instance, click tracking history, last log-in time, or links between people and assets, departments, and other people);
- Data that cannot be processed using rule-based approaches (e.g. encrypted data or blobs);
- Data that does not follow rules, as anything can be personal data, not just PII (e.g. contextual data, that is, data co-occurring with other data, entire comments or notes, entire

id	name	tel	socid	secret	status
0447921	Jane D.	0470555213	10force	297dbe7699dcfa6	Make the EU great again.

Table 2.1: Limitations of PII discovery

documents, and infinitely many other data types).

Moreover and, perhaps, more importantly, such solutions are not *identity-aware*.

- Personal data cannot be processed unless it is linked to the relevant legal basis and its rightful owner.
- Data subject requests cannot be granted unless the controller knows what data they have on a given individual, as well as its location.
- Policies (e.g. regarding retention periods) cannot be enforced, unless the data is linked to the data subject it belongs to.

Additionally, many of such approaches tend to overlook the effect of unstructured data and, more importantly, that of the information assets organisations collect, process and store during regular business activities, but generally fail to use for other purposes, also known as *dark data*[1].

SPECIAL aims to go beyond identifying essential business entities and personally identifiable information in structured data, and investigate alternative approaches to building data subject-centric digital enterprise inventories.

1.1 Identity-aware personal data discovery

Personal data being *any information relating to a data subject* requires context awareness, which is precisely why traditional rule-based approaches, today, are legally insufficient. In other words, rather than just trying to understand what type of data is stored in a data source, personal data discovery solutions should understand the nature of the data source, and the exact context a particular instance of data appears in.

The example given in Table 2.1 shows the typical limitations of PII discovery solutions. By ignoring the context, the data that cannot be processed using rule-based approaches (the *socid*, *secret*, and *status* attributes) will never be indexed.

1.1.1 Three axes of discovery

To outperform hardcoded rules, the discovery process would have to understand (that is, learn) what makes a relation, so it could infer what or who the data *relates* to. And given that a considerable share of enterprise data sources are relational (and, hence, structured) in nature, it is safe to assume that they can be a potential source of such reference knowledge.

Our approach starts with a blank slate, rather than a rule set, and uses sample (training) data to bootstrap the discovery process. A training dataset does not have to be very large, detailed, nor complete, but the provided data should uniquely identify at least one data subject.

[1]https://www.gartner.com/it-glossary/dark-data

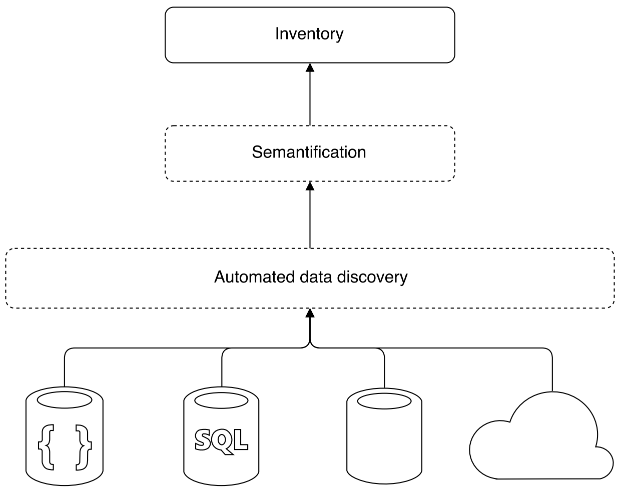

Figure 2.1: Assisted Data Discovery and Cataloging

- **X-axis (horizontal).** The discovery process starts by finding occurrences of the provided training data. As the system finds the data subject's data, it can also learn what is related to it and further extend the sample dataset by including the related data.
- **Y-axis (vertical).** It is worth noting that data profiling and PII discovery techniques, in certain cases, can still be valuable - for instance, to analyze the data vertically in the relevant columns, as well as the accompanying metadata. Where columns contain homogeneous data, labeling the sample data will potentially result in labeling the entire data assets/tables the data is contained in. That means that the data sample can then be expanded vertically to include other data subjects. The more the system discovers, the more accurate it becomes at its task.
- **Z-axis (unstructured).** Once enough reference knowledge has been acquired, the system can go beyond relational databases, to not only discover occurrences of personal data in unstructured data sources, but also link them to their respective owners.

1.2 Identities

As personal data is more often than not scattered across several systems, it is also not unusual to find the same piece of information in multiple data sources. Being able to identify and interlink all known sensitive data belonging to a subject, all occurrences of the same piece of information, and even the data that may not as yet have been accounted for, means being able to recreate the subject's virtual *identity graph*.

A complete graph can give an overview of what truly constitutes personal data, as well as help identify inconsistencies, remove duplicates and minimise risks pertaining to a potential data breach. It can also ensure that both the controller and the subject always have a clear picture of *what belongs to whom*, but also *what is correlated to what*, so individuals can exercise even more control over their own data. In the context of SPECIAL, this would translate into more detailed data lineage and enable fast and deep insights into data provenance and risk analysis.

The identity graph can be viewed as a map of the data within an enterprise; it describes the situation as it is. This can help inform data governance decisions and track progress, but, as such, these identity graphs do not change the underlying data. RDF provides the flexibility to represent such information in more than one way, but also the means to easily extend it further.

1.2.1 Identity graphs

In this section we describe the data model that we use to represent the identity of a data subject. The prefixes used in this, and other examples in this chapter are given in Listing 2.1. A data subject in this context means a natural person of whom we have registered data in our system. The identity is the collection of facts about that data subject.

Listing 2.1: Prefixes used in this chapter

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX tf: <http://xdc.tenforce.com/elements/>
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
```

We collect facts about identities in *identity graphs*. Such an identity graph contains the facts as well as their form, technical types and storage (Figure 2.2).

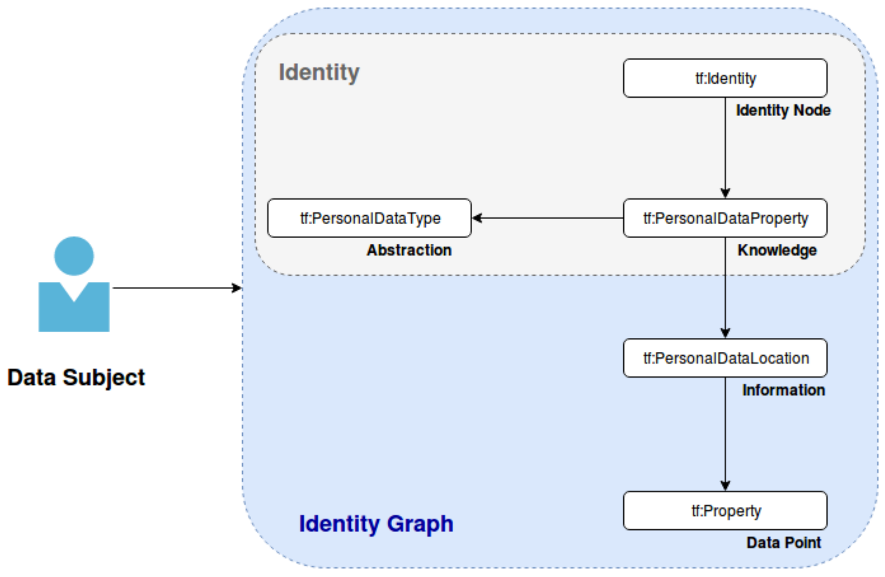

Figure 2.2: The concepts introduced in this section and their relation to the identity data model

For each data subject in the known set of data catalogs we build an identity graph. The center of the identity graph is an *identity* node. This identity graph is meant to describe every single piece of personal knowledge known about that data subject in a system.

We make a distinction between 3 different concepts concerning properties that, combined, make up an identity graph. These concepts are:

- *knowledge:* the canonical form of information;
- *abstraction:* the abstract notion of the information;

- *information:* actual information.

We will illustrate these terms with an example. We could state that in our system we have the following 2 data points: "Jane Doe" and "J. Doe". Both values are, in their canonical form, "Jane Doe". We refer to the data points and their values as information. The canonical form, "Jane Doe" in this case, will be referred to as knowledge. The abstract notion of this information would be 'a name'.

In the identity data model we will denote *knowledge* with *personal data property*. Since all *personal data properties* are singly linked to the *identity*, this results in a star shaped pattern (Figure 2.3).

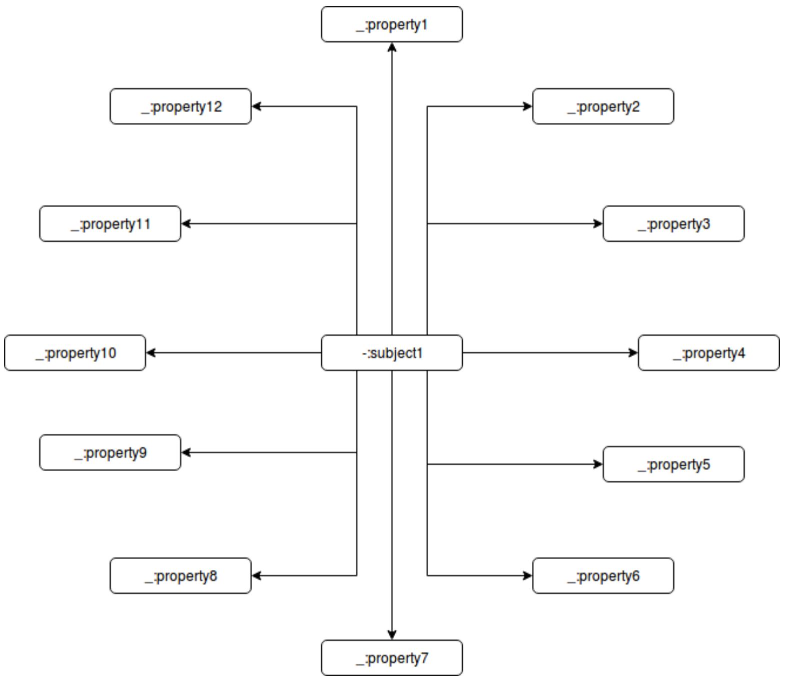

Figure 2.3: An "identity" together with it's *personal data properties* forming a star

The *abstraction* is captured in the *personal data type*. The *personal data type* links to a *skos:Concept* which is in scheme of a *skos:ConceptScheme* describing a *personal data taxonomy*. The *personal data type* also references the linked data attribute. The attribute in the example in this chapter could then be *foaf:name*.

Next, all the instances of this *information*, such as "J. Doe" and "Jane Doe" in the example, are linked to the *personal data property*. Those instances are indicated by a *personal data location*. Each of these 'personal data locations' holds a reference to it's storage and the value stored there. The reference to the storage holds sufficient information to identify a single point of data. We call this a *data point*.

A *data point* in this context is an atomic piece of data. This can be a field in a table, a textual part of a file, a cell in an excel file, and so on. An example of this in triples is given in Listing 2.2.

Listing 2.2: An identity with a personal data property in 2 different personal data locations as RDF

```
identities:Jane a tf:Identity ;
  tf:personalDataProperty properties:Name1 .

properties:Name1 a tf:PersonalDataProperty ;
  tf:personalDataType personalDataTypes:Name ;
  tf:personalDataLocation personalDataLocations:Location1 ;
  tf:personalDataLocation personalDataLocations:Location2 .

tf:personalDataTypes:Name a tf:personalDataType ;
  tf:refersToPredicate foaf:name ;
  tf:personalDataConcept concepts:Name .

personalDataLocations:Location1 a tf:PersonalDataLocation ;
  tf:storedIn datapoints:Datapoint1 ;
  tf:valueAtLocation "Jane␣Doe" .

personalDataLocations:Location2 a tf:PersonalDataLocation ;
  tf:storedIn datapoints:Datapoint2 ;
  tf:valueAtLocation "J.␣Doe" .
```

The same example as a diagram:

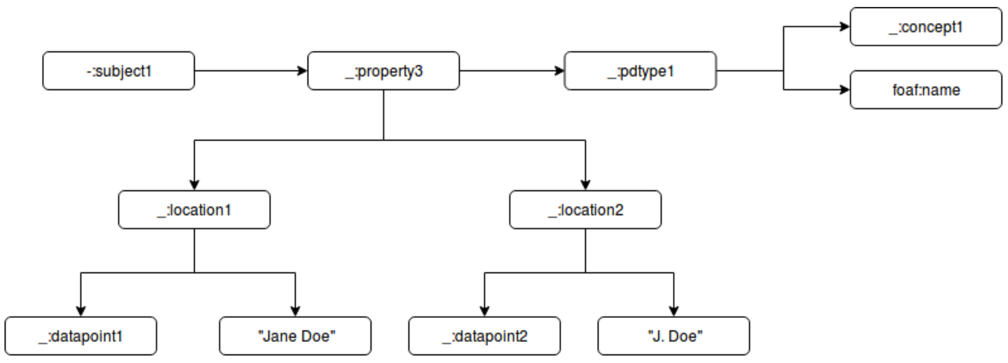

Figure 2.4: Illustrates the data in Listing 2.2

1.2.2 Shapes

As we continue exploring the data through our application, we maintain one or more aggregates. These aggregates capture the knowledge expressed by the body of identities. In a spatial sense they will contain the information that a typical *identity* within this system has a name and that the name can be found in that collection. Such an aggregation is called a *shape*.

Shapes are meant to be useful for exploring and for the discovery of previously unknown identities. Just like their identity counterparts, these shapes will form star-shaped graph patterns. Each shape has multiple pieces of knowledge. In contrast with the knowledge connected to an identity ('personal data properties'), the knowledge connected to a shape is not derived from an

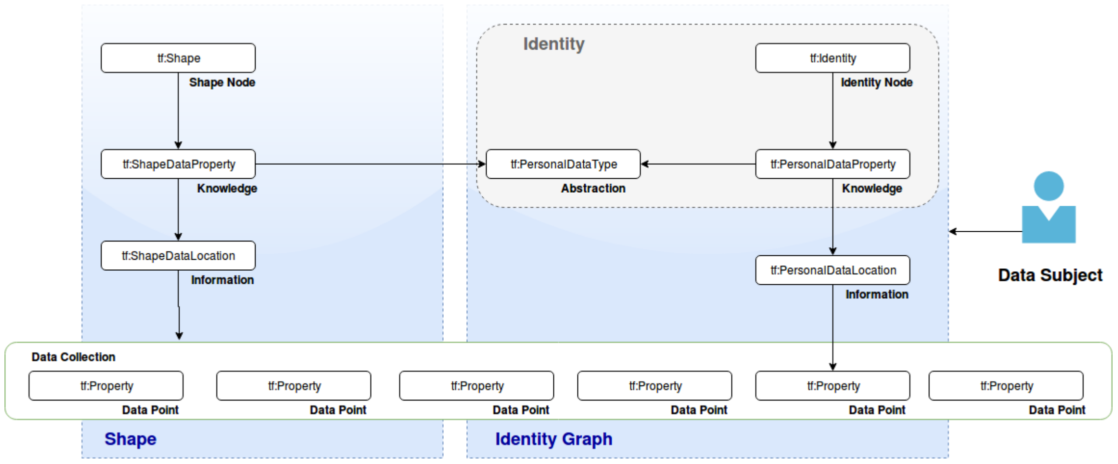

Figure 2.5: Illustrating the concepts of *shape* and how they relate to those of *identity*

actual piece of information such as "Jane Doe" or "J. Doe". The *shape data property* rather is an instance of a concept that connects a *shape data location* to a *personal data type*.

These *shape data locations* cannot point to actual cells, but instead to the column in a database table, the column in an excel spreadsheet, or the entire contents of a file. Such a *shape data location* then holds the information of the *data collection* to which it refers as well as a literal number indicating the number of occurrences of *personal data properties* which are of the *personal data type* to which this 'shape data property' belongs.

An example shape that is consistent with the previous identity graph example can be found in Listing 2.3.

Listing 2.3: A shape that has 1 shape property of type *personalDataTypes:Name* in triples

```
shapes:EmployeeShape a tf:Shape;
  tf:shapeDataProperty shapeProperties:Name2 .

shapeProperties:Name2 a tf:ShapeDataProperty ;
  tf:personalDataType personalDataTypes:Name ;
  tf:shapeDataLocation shapeDataLocations:Location3 .

shapeDataLocations:Location3 a ShapeDataLocation ;
  tf:storedIn datapointCollcetions:Collection1 ;
  tf:numberOfOccurences "1"^^xsd:int .
```

The complete data model is illustrated in Figure 2.5. Each discovered *personal data property* also includes a reference to the *shape data property* of which it is an instance.

1.3 Ontology Based Data Access

ETL approaches to collecting relevant data would inevitably introduce another layer of complexity, as well as additional data integrity and security risks. The derived identity graph mappings, however, could be used to create a virtualisation layer without replicating the original data, and provide limited access to authorised individuals on request.

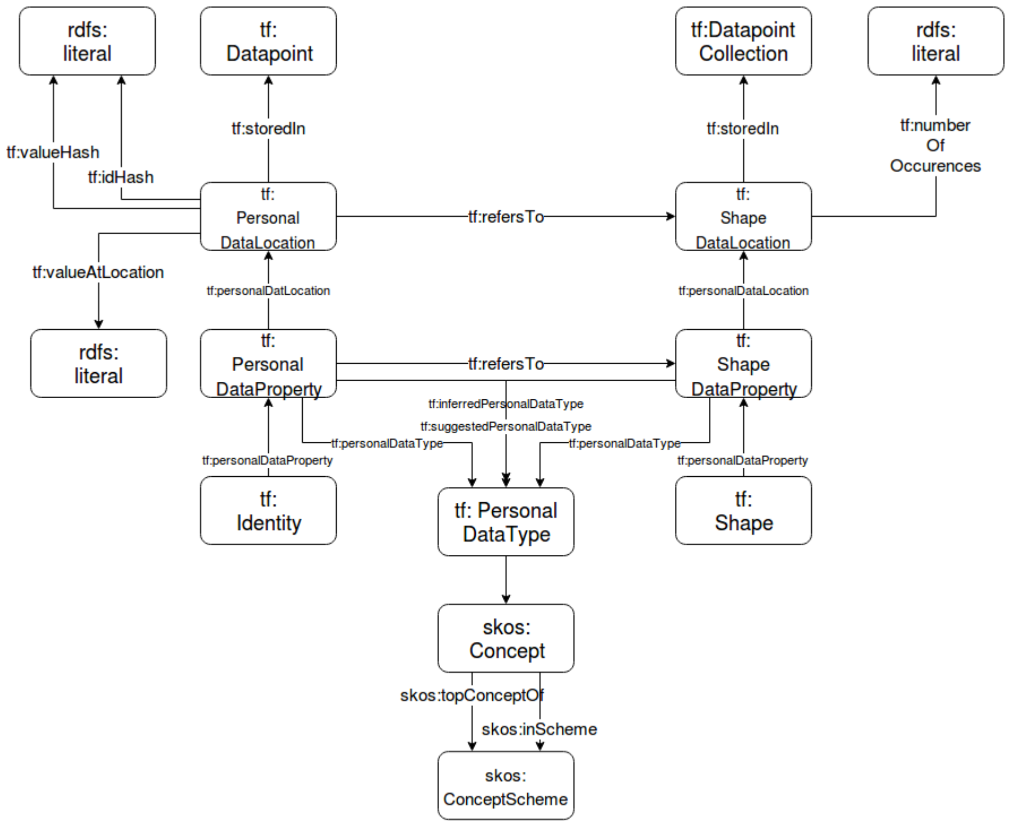

Figure 2.6: The identity graph model

R2RML[2] is a W3C-recommended language for expressing customised mappings from relational databases to RDF datasets. R2RML mappings create custom RDF views on top of existing relational data. The mappings themselves are RDF graphs, expressed in a structure and RDF vocabulary of choice, and encoded in Turtle syntax. An example mapping is given in *Listing* 2.4.

Listing 2.4: Example R2RML mapping

```
    rr:logicalTable [ rr:tableName "CUSTOMERS" ];
    rr:subjectMap [
        rr:template "http://example.com/person/{CUSTID}";
        rr:class foaf:Person;
    ];
    rr:predicateObjectMap [
        rr:predicate foaf:name;
        rr:objectMap [ rr:column "NAME" ];
    ];
    rr:predicateObjectMap [
        rr:predicate foaf:gender;
        rr:objectMap [ rr:column "GENDER" ];
    ].
```

[2]https://www.w3.org/TR/r2rml

As we will attempt to go beyond relational databases, the existing RDB2RDF solutions will not be sufficient. **RML**[3] is a superset of R2RML which can be used to map relational databases to the RDF data model. Instead of exclusively defining table names, RML can support any reference to any source within its Logical Source (rml:LogicalSource) which extends R2RML's Logical Table (rr:LogicalTable), thereby introducing support for a broader range of data formats. An example mapping of a JSON document to an RDF view is given in *Listing* 2.5.

Listing 2.5: Example RML mapping

```
    rml:logicalSource [
        rml:source "customers.json";
        rml:referenceFormulation ql:JSONPath;
        rml:iterator "$.[*].Person" ];
    rr:subjectMap [
        rr:template "http://example.com/person/{customer_id}";
        rr:class foaf:Person ];
    rr:predicateObjectMap [
        rr:predicate foaf:name;
        rr:objectMap [ rml:reference "name" ] ];
    rr:predicateObjectMap [
        rr:predicate foaf:gender;
        rr:objectMap [ rml:reference "gender" ] ].
```

Assuming the same data is present in the `CUSTOMERS` table and `customers.json` described above, both mappings would produce the same resulting graph. An example is shown in Listing 2.6.

Listing 2.6: Resulting RDF graph

```
ex:1234 a foaf:Person ;
    foaf:name "Jane␣Doe";
    foaf:gender "Female".
```

Although it may be seen as a data integration exercise, it is worth noting that, from an end-user perspective, a personal data inventory might not necessarily require direct/live data access, as it can also serve merely as a metadata repository. The extent to which the above solutions will be used is yet to be investigated.

2 Data Cataloging

Proper semantic lifting and interlinking of the above discussed personal data would also pave the way for automatic classification and cataloging (with little to no human intervention), thereby considerably minimizing the manual effort needed to maintain such information in fast-paced organisations.

Linking automatically classified and semantically annotated personal data to relevant business-level definitions would also introduce context, and even allow for ad-hoc recontextualisation of data. In addition to making the process of building and enforcing policies easier, the same approach holds the potential to be extended to a more general enterprise use case involving other types of data as part of a smarter Enterprise Metadata Management solution. This, however, calls for formal definitions of the relevant concepts and the relationships between them.

[3]http://rml.io

2.1 Catalog data model

We want to be able to express the concepts of (1) documentation, (2) ownership, (3) risk and (4) legal requirements with respect to the datasets stored in an enterprise system.

1. *Documentation* entails the general description, the characteristics and the limitations of a dataset.
2. *Ownership* means that we want to able to decide for each data point if it belongs to a data subject and if it is the case to which data subject it belongs.
3. *Risk* comprises of many factors. In essence it describes what types of data can be found within a dataset and the characteristics of its storage.
4. *Legal requirements* are coupled with ownership and risk. But also who the maintainer of the dataset is. And possibly historical information on breaches and how they were handled.

To be able to express the aforementioned concepts, it is key to capture the metadata about datasets as complete as possible. This section will describe how the meta information about the datasets are captured and stored in our catalog model (Figure 2.7).

2.1.1 DCAT-AP

The DCAT Application profile (DCAT-AP)[4] is a result of the European ISA^2 programme[5] and an extension of the Data Catalogue vocabulary (DCAT)[6], a W3C recommendation. Many public and private companies desire the use of open standards. Our catalog model builds on top of DCAT-AP to allow for interoperability and generalization beyond enterprise data management. Furthermore, using open standards such as DCAT-AP will help to avoid vendor lock-in.

The usage of this framework also helps us to clearly define the legal meaning of the terms supported by the vocabulary. Since there is an abundance of published data sets, amongst which there are datasets which are maintained by public organisations, there is ample example and clarification to the meaning of the terms used.

The dataset documentation covered by DCAT-AP suffices to cover our documentation needs. Next to the ability to describe a catalog and a dataset, it also supports data lineage and usage information. There is also support for classification through the use of themes.

Lastly, the usage of this standard will ensure a quality basis to extend our catalog model on.

2.1.2 Connections

We extend the DCAT model with *connections*. In contrast with DCAT *distributions*, which focus on the location of the dataset, connections describe all the needed information to make a connection to an enterprise datasource next to the *accessURL*. Those datasources have a multitude of attributes that vary based on the data source type. For instance, a Microsoft SQL Server database will require a username, a port, a password and a database name to be able to connect,

[4]https://joinup.ec.europa.eu/solution/dcat-application-profile-data-portals-europe
[5]https://ec.europa.eu/isa2
[6]https://www.w3.org/TR/vocab-dcat/

while for a public file server, the *accessURL* might suffice. Connections are also used to make an abstraction of the encoding.

The connection can also point to the server resource that can store a basic *security profile* of that server as well as information regarding its physical location.

2.1.3 Dataset profiles

The next extension to the DCAT-AP model is in the addition of a *dataset profile*. This profile will collect all meta information about a dataset. A dataset profile consists of several other profiles. At the moment, these other profiles are a *filter statistic* profile, a *schema profile*, a *natural language profile*, and a *count profile*. The idea is that the *dataset profile* can be extended with other profiles in the future.

- A *filter* is a simple function that gets a piece of data and returns true or false. Some filters will act on phone numbers, email addresses, person names, etc. You could imagine the person name filter to return *true* for "Jane Doe" while the phone number filter hopefully returns *false*. The filter statistic profile intends to capture the results of these filters for entire data collections.

- The *schema profile* describes the datasets schema. In a SQL database, tables will be mapped on classes (*tf:Class*) and columns will be mapped on properties (*tf:Property*).

- The *count profile* will relate to each class and describe the number of data points within that class.

- The *natural language profile* could be used to describe the suspected meaning of class names (column headers in a SQL table for instance).

2.1.4 Risk

When we discuss risk we are actually referring to a more complex notion. That notion is primarily conceived in 2 distinct scenario's.

The first of these scenario's is that somehow the data has been accessed and copied by a malevolent force outside of our system. The risk here is in third persons being able to identify a data subject. It could lead to the exposure of sensitive facts about said data subject.

The second scenario resides within our system borders. Whenever we process data, we need to be mindful of the need for consent for that processing. Consent implies attributal information about the data, but also about the purpose for its processing. As storage in itself is a form of processing we expect to have a legal base for that storage. The more sensitive a data point is, the more likely the need for a specific consent is. Consider, for example, the storing of a persons political opinion.

We reduce the aforementioned scenarios to 4 aspects of risk: *identification*, *sensitivity*, *storage* and *processing*. To define the risks associated with a certain dataset is, we use a compound figure. That is to be the sum of rating for the dataset for each of the 4 aspects.

The rating of *identification* is done by considering each data point within the dataset in itself. An email address or a national identification number are, for instance, strong identifiers, but so is the relation of this data point to other data points. An example of this is an IP address. In itself, the IP address is not sufficient to identify a data subject, but with the addition of access times it would allow the internet service provider to use the data points combined as an identifier.

Sensitivity is well described within the GDPR. Storage of sensitive data is considered to be a separate point of attention in the context of risk.

The third component in the assessement of risk is *storage*. By describing the physical storage solution in terms of geolocation and security, we build a storage score. The geolocation is rather simple as the GDPR itself has a geospatial component in its text. The security aspect of storage would include the use of certain physical server and network set ups.

The final component is the description of the types of *processing* the dataset is being used for, other than storage. This would involve both the actual processing as well as the location of the processing.

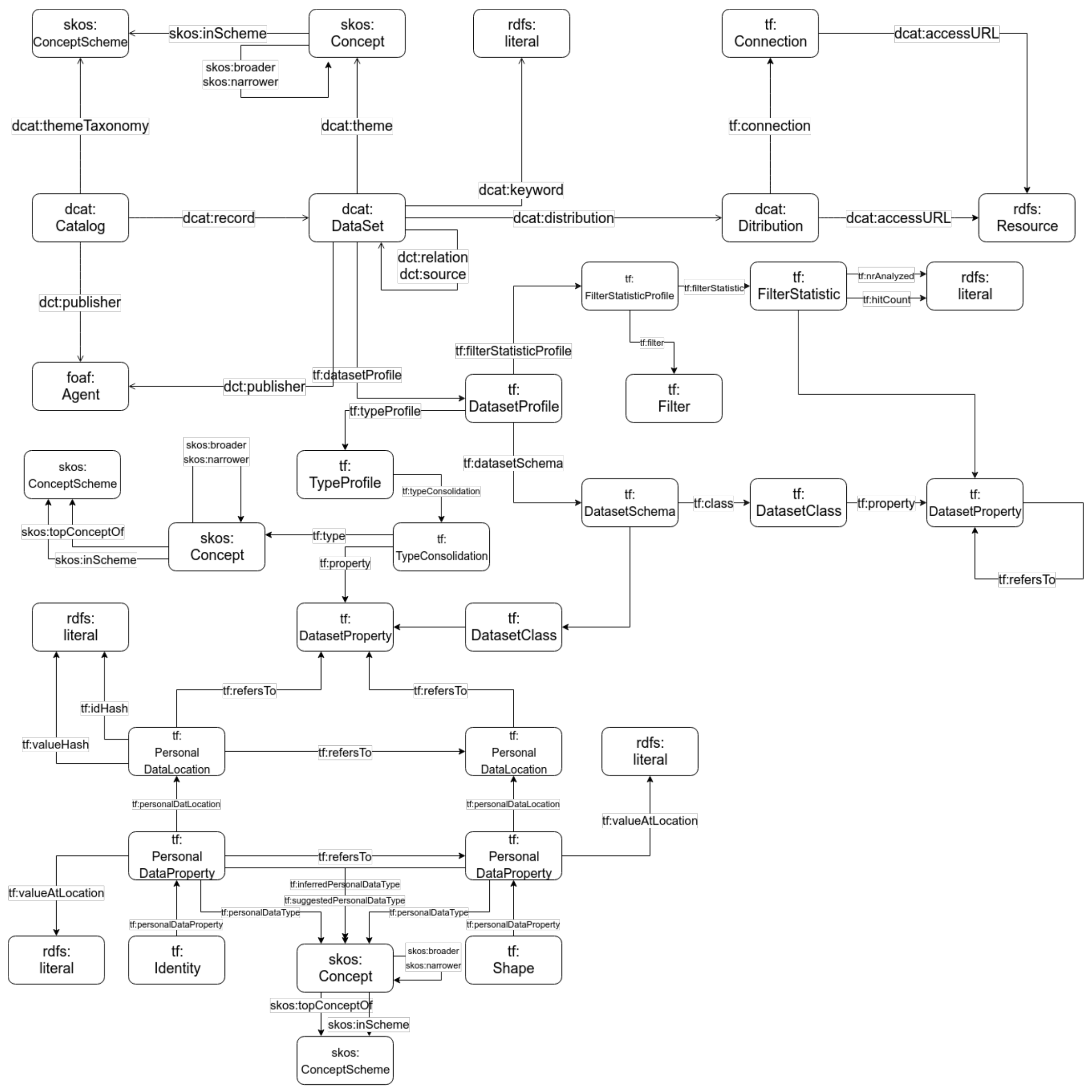

Figure 2.7: The data catalog data model

Chapter 3

Modelling the SPECIAL Policy Log

1 The Ledger

At the core of any transparency and compliance architecture is the logging of events in relation to the processing and sharing of personal data, as well as actions related to the consent provided (or revoked) by a data subject. The logs should be represented in a manner that enables verification that data processors abide by the access and usage control policies that have been associated with the data based on the data subject's consent and the applicable regulations.

In *D1.7 Policy, transparency and compliance guidelines V2 - Chapter 3* - (i) we identified the main requirements for the ledger (analogous to a "log") in order to provide transparency to data subjects, e.g the *completeness*, *interoperability* and *integrity* of the log, to name but a few, (ii) we outlined the main data to be captured so that the log can be used to automatically verify compliance with access and usage control policies specified by data subjects, i.e. we envisioned to maintain a record of all data processing and sharing events, (iii) we analysed the limitations of current ledgers, and (iv) we proposed to leverage the power of RDF and Linked Data to represent the events in the ledger in a machine readable manner.

Hereinafter, we focus on providing a concrete model to represent data processing and sharing events, including the consent provided by the data subject and subsequent changes to or revocation of said consent.

To do so, we provide a new vocabulary, referred to as *SPLog* (presented in the next section) that builds upon the *policy language* ontology presented in *Deliverable D2.1 Policy Language V1* and reuses well-known vocabularies such as PROV [9] to provide provenance metadata of the log.

By employing RDF/Linked Data technologies to represent the provenance events stored in the ledger we pursue the following contributions:

- Events are described in semantically unambiguous terms aligned to the same taxonomies defining usage policies, hence we facilitate automatic compliance checking. The first version of the compliance checking algorithm can be found in *Deliverable D2.8*.
- We set the basis to extend the event descriptions to cope with novel business, transparency and compliance needs, as vocabularies can be extended seamlessly.
- We support interoperability between ledgers thanks to RDF and Linked Data principles. An approach to integrate several ledges can be found in Section 2.8.

- Events can be grouped to facilitate scalability. Section 2.6 provides further information on such grouping mechanism. Note that we provide in *Deliverable D2.8* a big data architecture able to deal with the amount of information generated in the ledger.

- Events can be attached to an immutable record, potentially stored in a different ledger or knowledge base, to assure immutability. A discussion on potential immutable ledgers can be found in *D1.7 Policy, transparency and compliance guidelines V2*, while we discuss on using concrete immutable ledgers in *Deliverable D2.8*.

- Services providing transparency on top of the ledger can be built upon SPARQL [5] and simple inference mechanisms (as detailed in *Deliverable D2.8*).

Note that we make the following assumptions in this deliverable:

- We consider a log with monotone increasing size, disregarding *erasure*. Future implementations of the log can potentially make use of cryptographic deletion mechanisms in order to harmonise mandatory preservation requirements and the right to erasure. It is worth mentioning that a recent report produced by the Commission Nationale de l'Informatique et des Libertés (CNIL)[1] draws similar conclusions in a blockchain scenario.

- The content of the events could potentially be described at different granularities, from categorising the content in a simple taxonomy stating the type of data, processing, etc., involved in the event, to the most fine-grained description of the actual data associated to the event (e.g. concrete location of a data subject). On the one hand, this would allow companies to have flexibility with respect to the level of detail recorded in the log. On the other hand, if actual data is stored in the log (i.e. instances), the compliance checking mechanisms may need to perform a preprocessing step to infer the actual categories (classes) that should be verified against the consent provided by data subjects. This deliverable assumes the former case, i.e. log entries store categories (classes) such that compliance checking is based on class subsumption (as detailed in *Deliverable D2.8*). In any case, as companies may still want to store instance data associated to an event, e.g. in order to show the actual process data associated to an event in an integrated UI to data subjects, we provide an extension of the vocabulary to represent actual instance data (see Section 2.9).

- The log could potentially include the formalisation of the GDPR (see *D1.7 Policy, transparency and compliance guidelines V2* presenting initial guidelines) in order to integrate, in a single system, the log of the usage policies and the regulation policies in place. We design our model considering this potential integration, hence the taxonomy of log entries is flexible enough to accommodate the formalisation of the GDPR (in essence, they could be a specific type of "*Policy Entries*" with an initially fixed validity).

- Consents given by data subjects have a starting validity time and are defined forever unless a consent is replaced with a new consent (i.e. a consent replaces any previous consent). Updates and revocations are then implicit. Nonetheless we consider in this deliverable that companies may require to store a "*Consent Revocation*" action to simplify consent tracking and consent versioning.

[1]CNIL Blockchain report, https://www.cnil.fr/sites/default/files/atoms/files/la_blockchain.pdf

2 SPLog Vocabulary

This section introduces the **SPECIAL Policy Log Vocabulary (SPLog)**, a vocabulary to log data processing or sharing events (that should comply with a given privacy policy) as well as actions related to the consent provided (or revoked) by a data subject. Appendix 2.12.2 includes the vocabulary in OWL. We also provide concrete examples using the SPECIAL BeFit scenario of fitness tracking presented in *D1.7 Policy, transparency and compliance guidelines V2*.

2.1 Preliminaries

For designing the SPLog vocabulary we have carefully aligned with (i) the guidelines in *D1.7 Policy, transparency and compliance guidelines V2*, (ii) our policy language in *Deliverable D2.1 Policy Language V1*, (iii) the vocabulary and standard guidelines in *Deliverable 6.3 Plan for community group and standardisation contribution* and (iv) related work on log and event processing, with particular attention to the application in Linked Data scenarios.

In particular, we first followed (a) the large body of work in the Business Process Management (BPM) community that focuses on using process execution events for business process compliance monitoring [11] and process mining [16]. From this context, we borrow the following assumptions:

- We assume that a log entry contains data related to a *single process*. Thus, in our modelling, log entries are related to an uniquely identified process. For the sake of simplicity, we assume that data processing and sharing events correspond to exactly one *log entry*.

- In principle, events are instantaneous, thus they can be associated with a single timestamp. In addition, we decided to allow for grouping entries and include a duration (start and end timestamps) to favour scalability. For instance, a company may decide to insert one entry for each location gathered in the BeFit device. However this might result in scalability issues if the gathering rate is high. In contrast, the company can opt to register a log entry group for a *running activity*, hence the log entry group states that the position of the user was collected in a particular time frame.

- We integrated an optional *BPM* module in our model, in order to represent BPM information that might be present in the company and can complement the logging information. As such, each event in the log is related to a single process instance, typically called "*case*" in BPM. This in reflected in the model with a *Case* class. For example, a case could identify the daily routine of a user in our BeFit scenario. Events can be related to some activity. For example, an activity in BeFit could be "*userIsTraining*" which can be associated to several events. Nonetheless, not all logs and companies follow an organised BPM structure, hence we consider and encourage this distinction, although it is seen as optional for the model. Note that grouping the information according to BPM can help the data subject to discern about the concrete purpose of each individual action. For example, a process to perform a recommendation based on the training of data subjects can have an instance `recommendationUser1` that may consist of a set of events that *Collect* data, an event to *Analyse* data and a final event to *Recommend* something to the data subject. Section 2.12 shows how this minimum BPM vocabulary of processes, cases and activities can be extended to provide fine-grained details.

- We integrated an optional *Immutable* module to represent that a log entry can be additionally linked to its representation as an immutable record, potentially stored in a different ledger or knowledge base. A discussion on immutable ledgers can be found in *Deliverable D2.8*.

2.2 Conventions and namespaces

The namespace for the SPECIAL Policy Log Vocabulary is `http://www.specialprivacy.eu/langs/splog#`. We write triples in this document in the Turtle RDF syntax [12] using the following namespace prefixes:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX spl: <http://www.specialprivacy.eu/langs/usage-policy#>
PREFIX splog: <http://www.specialprivacy.eu/langs/splog#>
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
```

The key words *MAY*, *MUST*, and *SHOULD* are to be interpreted as described in [1].

2.3 Outline of the vocabulary

Figure 3.1 depicts an overview of the vocabulary. Several concepts and properties have been defined to cover the log and its entries, detailed in Section 2.4. In addition, the description of a policy log can be complemented with two optional conceptual modules (dashed), *BPM* and *Immutable*. As stated, BPM represents the optional BPM information from the company that can be attached to events, such as activities, cases and processes. In turn, an *Immutable Record* preserves a log entry, potentially in a different ledger or knowledge base (see Deliverable D2.8 for a discussion on immutable ledgers).

2.4 Concepts

In this section we first define the main concepts concerning the SPLog vocabulary. Then, we present a practical example where we consider the SPECIAL BeFit scenario of fitness tracking, introduced in *D1.7 Policy, transparency and compliance guidelines V2*.

2.4.1 Log

A log (represented as `splog:Log`) is a collection of data that records data processing and sharing events as well as consent-related activities (assertion and revocation). The data in a log can be described as belonging to one of the following categories:

Log metadata. This is metadata that *SHOULD* describe the log as a whole, such as the label or title, the software agent(s) it belongs to, etc. Metadata is described in Section 2.7. One of the most important aspects is the processor whose service is logged. This is modelled with the `splog:processor` property (a subproperty of `prov:agent`), relating the `splog:Log` and the corresponding `splog:Processor` instance.

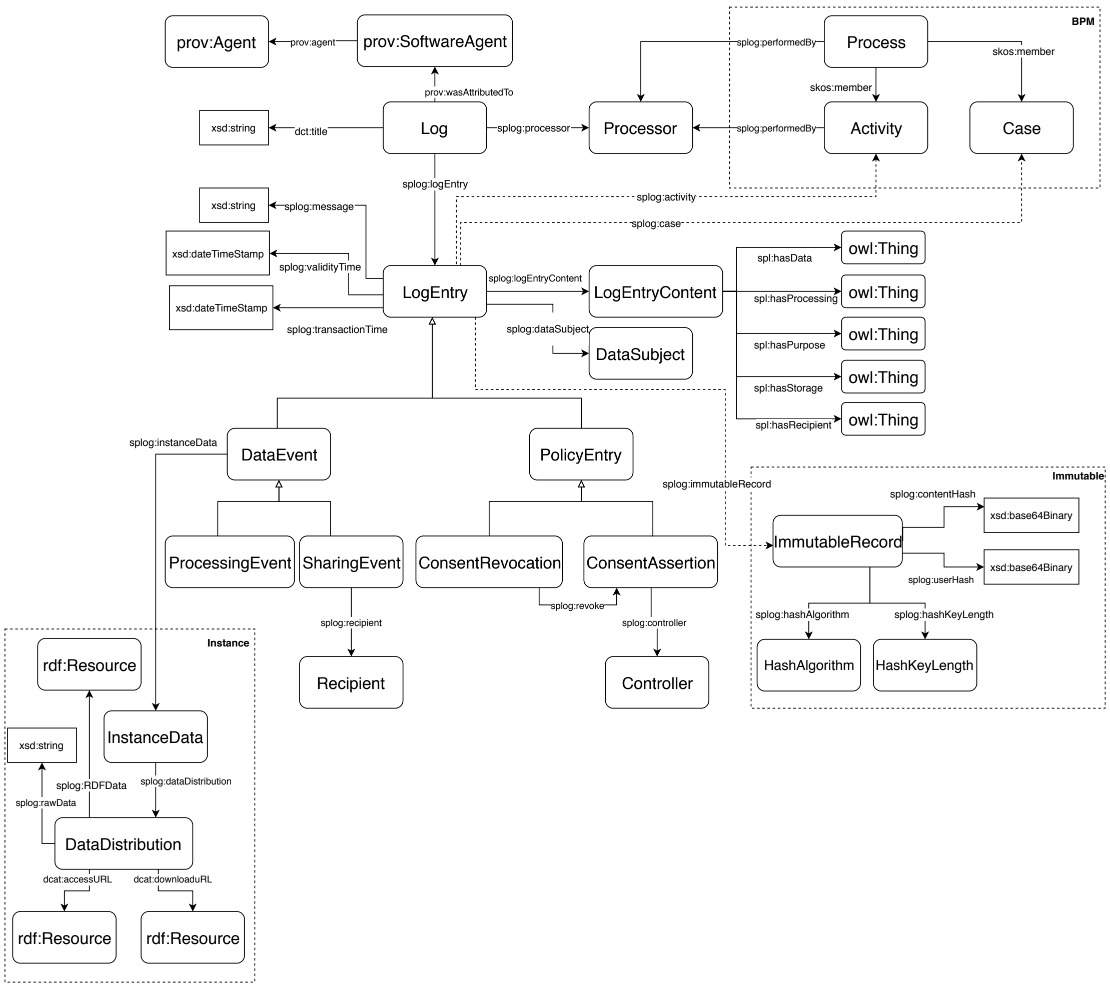

Figure 3.1: Pictorial summary of key terms and their relationship

Log Entries. This is the actual data contained in the log, represented with `splog:LogEntry`. The log *MUST* make use of the `splog:logEntry` property (a `prov:wasGeneratedBy` subproperty) to point to each entry in the log. Optionally, and for the sake of compactness, entries *MAY* be grouped into a given dimension or set of dimensions, conforming log entry groups. This is described in Section 2.6. In such a case, the log can point to the groups through the specific property `splog:logEntryGroup` (a subproperty of `splog:logEntry`).

2.4.2 Log entries

Log entries contain information about processing and sharing events associated with data subjects, as well as actions related to the consent provided (or revoked) by data subjects. These different types of entries are represented in our model with a classification of log entries, i.e., a hierarchy of classes, shown in Figure 3.1. Thus, a `splog:LogEntry` has two main types (subclasses), `splog:PolicyEntry` and `splog:DataEvent`, described as follows:

`PolicyEntry`. This class reflects log entries related to policies and consent. We currently consider two subclasses, `splog:ConsentAssertion` specifying a consent provided by a data subject to a `splog:Controller` (linked with a `splog:controller` property), and `splog:ConsentRevocation`, denoting the revocation of a given consent. Note that, as stated, we assume that a consent provided by a data subject replaces any previous consent, hence consent updates are implicit. Nonetheless, companies may wish to explicitly record a revocation entry pointing to the revoked consent, thus we include this capability via the `splog:revoke` property in our model. This latter may facilitate consent tracking and consent versioning.

`DataEvent`. This class considers log entries that are actually events on the data, i.e., the aforementioned data processing and sharing events. In the case of the latter, the concrete `splog:Recipient` instances can be specified, via the `splog:recipient` property.

In turn, the data in a log entry can be described as belonging to one of the following kinds:

Log entry metadata. This is metadata that *SHOULD* describe the entry as a whole. Metadata is described in Section 2.7.

Data subjects. The log entry *SHOULD* reference the data subject(s) involved in the entry. This is specified with the `splog:dataSubject` property (a `prov:wasAssociatedWith` subproperty) pointing to the appropriate `splog:DataSubject` involved in the entry. For the sake of simplicity, we assume that an entry is related to a single data subject, but multiple data subjects *MAY* be specified using multiple `splog:dataSubject` properties. Note that in case of anonymised logs, no subject can be specified.

Content. The log entry *MUST* reference the actual data of the log. This is specified with the `splog:logEntryContent` property, which points to the appropriate instance of `splog:LogEntryContent`. This is described in Section 2.5.

Timestamps. The log entry *MUST* reference the time at which the event occurred using the `splog:validityTime` property (subproperty of `prov:atTime`). Note that this is based on the aforementioned assumption of representing instantaneous events. For the sake of log preservation, the log entry *SHOULD* also reflect the time in which the log was recorded, using `splog:transactionTime` (a `dct:issued` subproperty).

Message. The log entry *SHOULD* reference a `splog:message` of the log representing a human-friendly text.

InmutableRecord. The entry *MAY* reference a `splog:InmutableRecord` of its contents.

Activities. The log entry *MAY* reference the concrete BPM `splog:Activity` and the BPM `splog:Case`, via the `splog:activity` and `splog:case` properties, respectively. Activities and cases are members of a `splog:Process`, specified with `skos:member`, and they can point to the involved `splog:Processor`, via `splog:performedBy`. Further information on how this information can be enriched to provide further BPM information can be found in Section 2.12.

DataInstance. The `splog:DataEvent` log entries *MAY* reference the concrete instance data, `splog:InstanceData` (subclass of `dcat:Dataset`) associated to an event, via the `splog:instanceData` property. The instance data can be materialized in one or more `splog:DataDistribution` (subclass of `dcat:Distribution`). A distribution can contain the raw data itself (`splog:rawData`) or `splog:downloadURL` (subproperty of `dcat:downloadURL`), typically described via a dct:format media type. If the data cannot be directly downloaded but there is an access point (A landing page, feed, SPARQL endpoint), the URL can be specified with `splog:accessURL` (subproperty of `dcat:accessURL`). Additionally, if the data is RDF itself, the concrete resource (e.g. an RDF resource or named graph) can be specified via `splog:RDFData`. Further information on concrete uses is provided in Section 2.9.

2.4.3 Examples

The following example provides a quick overview of how the SPECIAL Policy Log vocabulary might be used to represent a log. We make use of our BeFit scenario: we assume (i) Sue is using a wearable appliance for fitness tracking from BeFit, (ii) the application is tracking the location of Sue for *health* purposes, (iii) a new location is stored in a particular database (called *BeFitDatabaseEurope*) and reflected in the log (called *BeFitLog*). Let us also assume that the namespace for the BeFit company is `befit:` (pointing to the appropriate IRI), being `befit:Us` the main reference of the company. We first show the general log description in Listing 3.1.

Listing 3.1: Log description for BeFit devices

```
befit:BeFitLog a splog:Log;
    dct:title               "Log of BeFitDatabaseEurope"@en;
    dct:description         "This contains events on BeFitDatabaseEurope
                             tracking devices geo-located in Europe"@en;
    dct:issued              "2018-02-14"^^xsd:date;
    prov:wasAttributedTo    befit:BeFitDatabaseEurope ;
    splog:processor         befit:Us .
```

Then, we include a new entry in the log, which is a processing event (uniquely identified as `befit:entry3918`) referencing a new tracking position of Sue, shown in Listing 3.2. We assume Sue's unique identifier is `befit:Sue`, which of course, as all the log information can be kept internal to the company. The collection of the new position took place on the 3rd of January, 2018, at 13:20 (i.e. validity time) and the event was recorded few seconds later (i.e. transaction time).

Listing 3.2: A new event for Sue's BeFit device

```
befit:BeFitLog splog:event befit:entry3918 .

befit:entry3918 a splog:ProcessingEvent;
    dct:title               "Collection of new device position"@en;
    splog:dataSubject   befit:Sue ;
    dct:description         "We collected a new position of your BeFit
                             device in our database in Europe"@en;
    splog:transactionTime   "2018-01-10T13:20:50Z"^^xsd:dateTimeStamp;
    splog:validityTime      "2018-01-10T13:20:00Z"^^xsd:dateTimeStamp;
    splog:message           "Tracking position by GPS... collected!" ;
    splog:eventContent      befit:content3918 ;
    splog:inmutableRecord   befit:iRec3918 .
```

Note that, in the previous description, the log entry `befit:entry3918` is an instance of a `ProcessingEvent`, `befit:iRec3918` represents the immutable version of the event (described below), and `befit:content3918` points to the actual content of the event, defined in the following Listing 3.3.

Listing 3.3: The content of a new event for Sue's BeFit device

```
befit:content3918 a splog:LogEntryContent;
    dct:description      "This contains the data collected by a BeFit
                         device on January 2018 in Vienna, only for
                         the health purpose of the service"@en;
    spl:hasData          svd:Location;
    spl:hasProcessing    befit:SensorGathering;
    spl:hasPurpose       befit:HealthTracking;
    spl:hasStorage       [has:location svl:OurServers];
    spl:hasRecipient     [a svr:Ours].

befit:SensorGathering rdfs:subClassOf svpr:Collect .
befit:HealthTracking rdfs:subClassOf svpu:Health .
```

In turn, the immutable record can be defined as the hash of the content and the data subject, which can be kept in a different ledger or knowledge base, together with the definition of the hash algorithm. A simple example is shown in Listing 3.4.

Listing 3.4: A new event for Sue's BeFit device

```
befit:iRec3918    a splog:InmutableRecord ;
            splog:hashContent "AZ8QWE..."^^xsd:base64Binary ;
            splog:hashUser    "BHJQQ..."^^xsd:base64Binary ;
            splog:hashAlgorithm eg:hashRSA ;
            splog:hashKeyLength eg:hash2048 .
```

2.5 Log entry content

The log entry content is represented by the `splog:LogEntryContent` class, which is a type of (`rdfs:subClassOf`) the SPECIAL `spl:Authorization` defined in *Deliverable D2.1 Policy Language V1*. This way, event content and data policy authorisations can be checked for compliance. Nonetheless, note that the concept *spl:Authorization* can be confusing in this scenario as this can refer to a policy or an actual *operation* reflected in the log. Thus, we are planning to rename the `spl:Authorization` class in the new versions of the policy ontology, e.g. using `DataUsage` with the understanding that a set of `DataUsage` instances can either be policies (i.e. authorisations) or actual operations reflected in the log.

The `splog:LogEntryContent` class definition *MUST* include the five elements defined in the SPECIAL usage policy language (see *Deliverable D2.1 Policy Language V1*):

spl:hasData. It specifies the data involved in the event.

spl:hasProcessing. It specifies specifies how is data processed.

spl:hasPurpose. It specifies the purpose of the data processing.

spl:hasStorage. It specifies where and for how long is the data stored.

spl:hasRecipient. It specifies potential disclosures to other recipients, including third parties.

Further information on compliance checking between the log entry content and the consent provided by the data subject can be found in Deliverable D2.8.

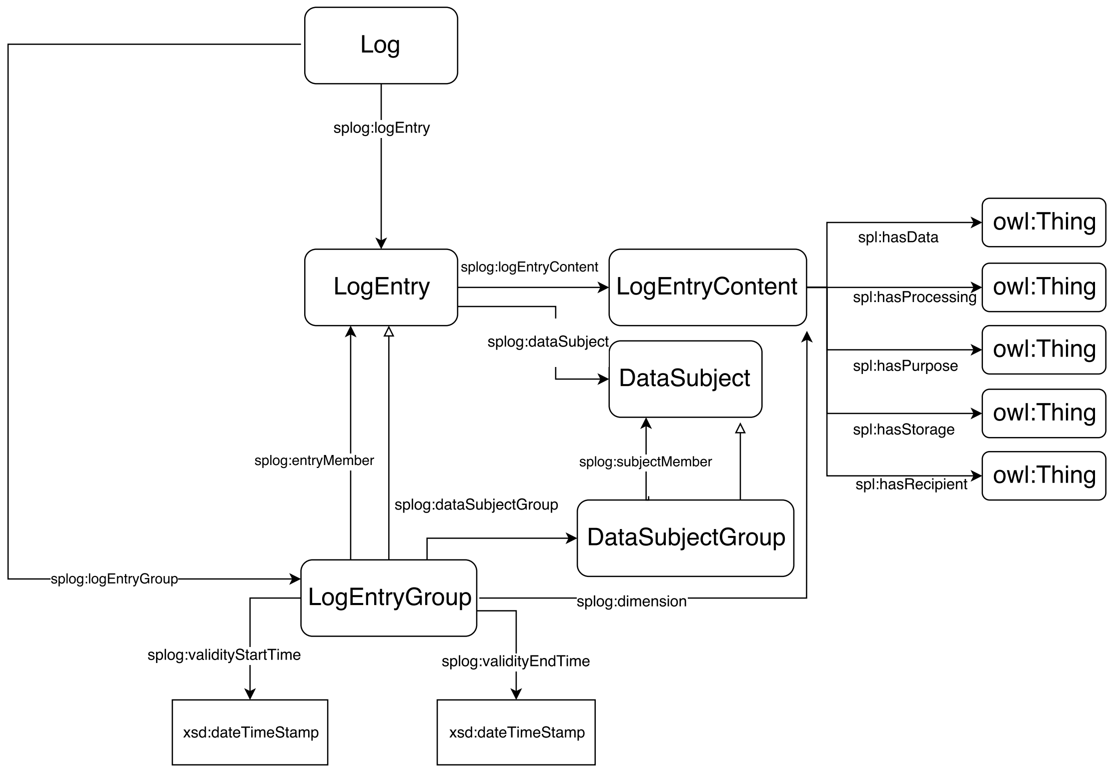

Figure 3.2: Pictorial summary of log entry grouping

2.6 Grouping log entries

Log entries, and in particular data processing and sharing events, are meant to provide fine-grained descriptions, typically concerning a single data subject and action (i.e. a single data processing and sharing event). Thus, in those scenarios where there exists a continuous flow of information, such as the envisioned big data application, log processing can quickly suffer from scalability issues when it is needed to serve both transparency and compliance purposes. The scalable, big data infrastructure proposed in *Deliverable D2.8* is aimed at coping with such scenarios. In addition, we envision grouping mechanisms to group, slice and dice log entries in order to support presentation and processing. In the following, we present the `splog:LogEntryGroup` concept.

A log entry group is a subclass of a log entry, containing information about one or more log entries. For instance, in our BeFit use case, a log entry group could be used to represent (as a single entry) the collection of data during a running activity of a data subject in BeFit. Another scenario regards grouping information for several data subjects, e.g. to group the act of providing a recommendation to several subjects. Figure 3.2 shows and overview of the main components for grouping.

The data in a group can be described as belonging to one of the following kinds:

Log entry group metadata. This is metadata that describes the log group as a whole. Metadata is described in Section 2.7.

Timestamps. The entry *MAY* reference the time interval considered in the group, using the `splog:validityStartTime` and `splog:validityEndTime` properties (subproperties of `prov:startedAtTime` and `prov:endedAtTime`, respectively), denoting the validity time. Similarly to log entries, a log entry group SHOULD also reflect the time in which the entry was recorded using `splog:transactionTime` (a `dct:issued` subproperty).

Dimensions. The group *MUST* reference the component(s) it groups. This is specified with the `splog:dimension` property (a subproperty of `splog:logEntryContent`), pointing to a particular `splog:LogEntryContent`.

Data subject. The group *MAY* reference the data subject(s) it groups, using the property `splog:dataSubjectGroup` (`prov:wasAssociatedWith` subproperty). This property points to a `splog:DataSubjectGroup` instance that groups all the data subject members in the group via `splog:subjectMember` (a `skos:member` subproperty).

Entries. The group *MAY* point to the particular entries included in the group through the `splog:entryMember` property (a `skos:member` subproperty). This can serve the traceability requirement identified in *D1.7 Policy, transparency and compliance guidelines V2* - *Chapter 3* -, stating that it should be possible to know about any previous processing of the data and link events in a manner that supports traceability of processing.

The following example in Listing 3.5 shows a log grouping the category of recommendations given to Sue, John and Rose during a month.

Listing 3.5: A grouping example merging all recommendations given in a month

```
befit:BeFitLog a splog:Log ;
    splog:logEntryGroup befit:recommendationsJanuary2018 .

befit:recommendationsJanuary2018 a splog:logEntryGroup
    splog:transactionTime    "2018-02-01T00:05:00Z"^^xsd:dateTimeStamp;
    splog:validityTime       "2018-01-31T23:59:59Z"^^xsd:dateTimeStamp;
    splog:dataSubjectGroup    befit:basicSubjectGroup;
    splog:dimension           befit:templateOfferRecommendation .

befit:basicSubjectGroup splog:member befit:Sue, befit:John, befit:Rose.

befit:templateOfferRecommendation a splog:LogEntryContent ;
    spl:hasData               befit:OfferRecommendation;
    spl:hasProcessing         befit:MonthlyDataAnalysis;
    spl:hasPurpose            befit:MonthlyOffersRecommendation;
    spl:hasStorage            [has:location svl:OurServers];
    spl:hasRecipient          [a svr:Ours].

befit:OfferRecommendation rdfs:subClassOf svd:Location;
    rdfs:comment  "We recommended you an offer at the end of the month
                   based on the location of your device during the
                   given month. The concrete offer is not stored in
                   this log" .

befit:MonthlyDataAnalysis rdfs:subClassOf svpr:Analyze .
befit:MonthlyOffersRecommendation rdfs:subClassOf
   befit:RecommendationActivity .
befit:RecommendationActivity rdfs:subClassOf svpu:Marketing .
```

2.7 Log metadata

Logs, log entries and log entry groups *SHOULD* be marked up with metadata to support presentation and processing. Dublin Core Terms [4] *SHOULD* be used for representing the key metadata annotations commonly needed for Logs. The recommend minimum core set of metadata terms is:

- `dct:title` - may be same as `rdfs:label`
- `dct:description` - may be same as `rdfs:comment`
- `dct:issued` and `dct:modified` - may specify additional times.

Additional metadata terms can be used for describing policy logs, which is one of the goals of the *W3C Data Privacy Vocabularies and Controls Community Group*[2].

2.8 Provenance information

In the log model, we assume that the description of entries coming from different systems can be merged and integrated together in a single store, which will potentially serve transparency and compliance mechanisms.

In certain scenarios, named graphs can be used to encapsulate logs before integrating entries coming from different subsystems. For example, let us assume a gym company "ViennaGym", referred to with the namespace `viennagym`, makes offers to Sue based on a mutual sharing policy with BeFit. Listing 3.6 builds upon the previous gathering event (see Listing 3.2) and shows the integration with a marketing event from ViennaGym providing offers to Sue. First, the data item is gathered by BeFit (previous example), then it is shared between BeFit and ViennaGym, and finally this latter uses the data to provide marketing advertising. These series of events are encapsulated in three graphs `befit:tracking`, `befit:sharing`, and `viennagym:marketing` respectively. We make use of the TriG [3] syntax to extend Turtle with named graphs.

[2]https://www.w3.org/community/dpvcg

Listing 3.6: Example of integrating several events for Sue's BeFit device using provenance information in named graphs

```
# The default graph may include metadata about the graphs

befit:tracking prov:agent befit:Us .
befit:sharing prov:agent befit:Us .
viennagym:marketing prov:agent viennagym:Us .

# The following graph encodes information gathered from BeFit devices
befit:tracking{

  befit:entry3918 a splog:ProcessingEvent;
     prov:wasAssociatedWith befit:Sue ;
     splog:transactionTime  "2018-01-10T13:20:50Z"^^xsd:dateTimeStamp;
     splog:validityTime     "2018-01-10T13:20:00Z"^^xsd:dateTimeStamp;
     splog:message          "Tracking position by GPS.. collected!" ;
     # ... other metadata ...
     splog:content          befit:content3918 .

  befit:content3918 a splog:LogEntryContent;
     spl:hasData          svd:Location;
     spl:hasProcessing    befit:SensorGathering;
     spl:hasPurpose       befit:HealthTracking;
     spl:hasStorage       [has:location svl:OurServers];
     spl:hasRecipient     [a svr:Ours].

  # ... other descriptions ...

}

# This graph encodes sharing events between BeFit and ViennaGym
befit:sharing{

  befit:entry4253 a splog:SharingEvent;
     prov:wasAssociatedWith befit:Sue ;
     splog:transactionTime  "2018-01-15T09:02:30Z"^^xsd:dateTimeStamp;
     splog:validityTime     "2018-01-15T09:00:00Z"^^xsd:dateTimeStamp;
     splog:message          "Sharing GPS positions with a partner" ;
     # ... other metadata ...
     splog:recipient        viennagym:Us ;
     splog:content          befit:content4253 .

  befit:content4253 a splog:LogEntryContent;
     spl:hasData          svd:Location;
     spl:hasProcessing    befit:SecureTransferPartner;
     spl:hasPurpose       befit:BefitpartnerRecommendation;
     spl:hasStorage       [has:location svl:OurServers];
     spl:hasRecipient     viennagym:Company.


  befit:SecureTransferPartner rdfs:subClassOf svpr:Transfer .
  befit:PartnerRecommendation rdfs:subClassOf
     befit:RecommendationActivity .
  befit:RecommendationActivity rdfs:subClassOf svpu:Marketing .
  viennagym:Company rdfs:subClassOf spl:AnyRecipient .
}
```

```
# This graph encodes the marketing information of Sue by ViennaGym
viennagym:marketing{

viennagym:entry1111 a splog:ProcessingEvent;
     prov:wasAssociatedWith befit:Sue ;
     splog:transactionTime  "2018-01-27T13:00:30Z"^^xsd:dateTimeStamp;
     splog:validityTime     "2018-01-27T13:00:00Z"^^xsd:dateTimeStamp;
     splog:message          "Send offer of our gym!" ;
     # ... other metadata ...
     splog:content           viennagym:marketing6590 .

 viennagym:marketing6590 a splog:LogEntryContent;
      spl:hasData           svd:Location;
      spl:hasProcessing     viennagym:Analysis;
      spl:hasPurpose        viennagym:GymRecommendation;
      spl:hasStorage        [has:location svl:OurServers];
      spl:hasRecipient      [a svr:Ours].

 viennagym:Analysis rdfs:subClassOf svpr:Analyze .
 viennagym:GymRecommendation rdfs:subClassOf svpu:Marketing .

}
```

2.9 Recording instance data

In principle, the main objective of the SPLog is to record data processing and sharing events, together with policy-related events (consent assertion and revocation), keeping the actual (instance) subjects' data in a different ledger. However, as we describe in Section 2.4.2, SPLog additionally provides an optional *instance* module (a) to store such instance data, or (b) to refer to (a service or API) where the instance data can be located. In the following, we provide details and examples on the potential use of this vocabulary for these two cases. In both cases, we consider a `splog:InstanceData` class (a subclass of `dcat:Dataset`) associated to a `splog:DataEvent` log entry via the `splog:instanceData` property, as explained in Section 2.4.2. Then, the instance data can be served in different `splog:DataDistribution` (subclass of `dcat:Distribution`), e.g. one distribution stored in raw CSV data and one in JSON data. Combining storing and referenced data is also possible.

2.10 Storing instance data

A first possibility is to store the actual data in the log. For instance, BeFit may decide to store in the log both the data collection event and (a copy of) the actual collected data of Sue's Befit device. Note that physically storing the instance data in the log implies that the log contains (even more) sensitive data. Thus, in general, it is not recommended that the instance data are kept on a public ledger (such as blockchain), as a security breach or a future hash break would expose the actual data. In addition, similarly to the previous case of the log entries, an immutable ledger would prevent the controller of deleting or rectifying the data (as it is required), hence cryptographic deletion mechanisms must be in place for the actual data.

The SPLog vocabulary provides two ways of storing instance data, (a) storing raw data (e.g. JSON, CSV, etc.) or (b) storing the semantic representation of the data (i.e. RDF data).

Storing raw data. In this case, the `splog:DataDistribution` contains the raw data itself, using the `splog:rawData` property and further described with additional properties such as `dct:format` media type or `dcat:byteSize`.

The following example in Listing 3.7 shows an example of a raw distribution in CSV, showing the collection of location data from Sue.

Listing 3.7: An example of storing raw data (CSV)

```
befit:entry3918 a splog:ProcessingEvent;
    # ... other metadata ...
    splog:instanceData befit:instance3918 .

 befit:instance3918 a splog:IntanceData;
    dct:title "Actual collected data";
    dcat:contactPoint befit:CollectionContactPoint ;
        splog:dataDistribution befit:distribution3918_1 .
    # ... other descriptions ...

 befit:distribution3918_1 a splog:DataDistribution;
    a splog:DataDistribution;
    dcat:mediaType "text/csv" ;
    dcat:byteSize "304"^^xsd:decimal ;
    splog:rawData "PersonName,Position,Time \n Sue,48.2082 N, 16.3738 E,
                    2018-01-27T13:00:00Z" .
```

Storing RDF data. In case the data is actually RDF data, the concrete resource (e.g. an RDF resource or named graph) can be specified via `splog:RDFData`.

The following example in Listing 3.8 shows an example of storing a distribution in RDF, showing the collection of heart rate data from Sue.

Listing 3.8: An example of storing RDF data

```
 befit:instance5000 a splog:IntanceData;
    dct:title "Actual collected data";
    dcat:contactPoint befit:CollectionContactPoint ;
        splog:dataDistribution befit:distribution5000_1 .
    # ... other descriptions ...

 befit:distribution5000_1 a splog:DataDistribution;
    a splog:DataDistribution;
    splog:RDFData befit:collection_5000 .

 befit:collection_5000 foaf:name "Sue" ;
    befit:heartRate 80 .
```

Note that SPLog do not restrict the specific RDF vocabulary, hence processors can also make use of the data catalog vocabulary presented in the previous section.

In addition, existing mechanisms to produce and represent an encrypted, or partially encrypted, RDF graph can be used [6, 7]. The following example in Listing 3.9shows a potential encrypted scenario using the crypto ontology [6].

Listing 3.9: An example of storing encrypted RDF data

```
befit:distribution66_1 a splog:DataDistribution;
   a splog:DataDistribution;
   splog:RDFData befit:collection_enc_66 .

befit:collection_enc_66 crypto:equal "zhk....kjg" ;
   crypto:keyLength crypto:2048 ;
   crypto:algorithm crypto:rsa .
```

2.11 Referring to the location of the instance data

In this particular case, the instance data is located externally. Note that a first possibility is that the distribution makes use of the aforementioned `splog:RDFData` property, but the resource itself is external. In that case, the data can be retrieved with a standard Linked Data dereferenciation.

In general, as explained, SPLog provides access to external data via `splog:downloadURL` (subproperty of `dcat:downloadURL`), typically described via a dct:format media type, or `splog:accessURL` (subproperty of `dcat:accessURL`), when the data cannot be directly downloaded but there is an access point (e.g. landing page, feed, SPARQL endpoint).

The following example in Listing 3.10 shows an example of a reference to a JSON storing the collection of heart rate data from Sue at the given time.

Listing 3.10: An example of referring to external JSON

```
befit:instance888 a splog:IntanceData;
   dct:title "Actual collected data";
   dcat:contactPoint befit:CollectionContactPoint ;
       splog:dataDistribution befit:distribution888_1 .
   # ... other descriptions ...

befit:distribution888_1 a splog:DataDistribution;
   a splog:DataDistribution;
   dcat:mediaType "application/json" ;
   dcat:downloadURL
    <http:example.org/internalAPI/getHistoricData/Sue/20180325090930>.
```

It is worth noting that companies should be responsible for ensuring a “permanent URI” to the external data, and to respect the consistency such that the reference can fetch the actual data at the given point of time. Potential implementation can consider the use of history tables to keep the full history of changes, or the use of the Memento protocol [14] to reflect the resource versioning and to access/time-negotiate the version at a given datetime. Examples of the latter can be found in the W3C standard “Data on the Web Best Practices” [10].

2.12 Extended BPM information

As described in Section 2.4.2, the SPECIAL Policy Log vocabulary provides an optional BPM module that includes a minimum vocabulary to describe BPM information attached to events, such as BPM processes (`splog:Process`), cases (`splog:Case`) and activities (`splog:Activity`). This minimum vocabulary emerged from the BPM interest and requirements of our SPECIAL use cases, and it should cover a wide range of scenarios dealing with BPM. Note, however, that this vocabulary can be further extended to consider more complex BPM-guided scenarios. In the following, we briefly review two additional extension points as a guideline for future implementations.

2.12.1 Process mining and the IEEE standard eXtensible Event Stream (XES)

Process mining is a discipline whose aim is to discover, monitor and improve real processes by extracting knowledge from event logs [15], which represents actual processes in an organization. Such event logs, in practice, can take a plethora of different forms and formats, which hampers the analysis process. To tackle this issue, the IEEE standard eXtensible Event Stream (XES) proposes an XML-based standard for event logs, hence providing a well-established interchange format of event log data between tools and application domains [8].

The basic elements of an XES document are (i) the *log* object, which contains information about a specific process (e.g. collecting information from BeFit devices), (ii) the *trace* objects, describing the execution of one specific case of the logged object (e.g. one specific collection of data from Sue's BeFit device), (iii) *events* belonging to a particular trace (e.g. Sue's heartrate is recorded), and (iv) attributes describing logs, traces and events (e.g. the user ID). In addition, XES introduces the concept of *event classifiers* to assign each entity an identity, and *extensions*, in order to define and use a set of attributes (i.e. a reusable template).

Recently, Calvanese et al. [2] propose a XES event ontology to represent such XES documents, together with OBDA tools (e.g. *ontprom*[3]) to annotate and extract event logs from relational databases.

Thus, a particular instance of the SPECIAL log can be further enriched with references to XES information in those companies with a strong BPM focus, with production tools already managing XES documents or implementing OBDA solutions over existing databases.

Figure 3.3 shows the proposed connection (via `rdfs:seeAlso`) between the SPECIAL Policy Log vocabulary, with the original BPM module, and the elements of the aforementioned XES event ontology [2]. In our model, `splog:Log` can refer to concrete `xes:Log` elements, `splog:Case` refers to a particular `xes:Trace` and each `splot:LogEntry` can then refer to a `xes:Event`. Although, by default, we do not create references to attributes in XES, it is worth noting that each metadata property of an event (e.g. the validityTime) can be collected in a XES attribute an even grouped in a XES extension.

We expect these references can provide (i) an integration to established BPM processes in a company, (ii) further insights on the process behind each event in the log, and, conversely, (iii) an integration to perform process mining over events in relation to the processing and sharing of personal data.

2.12.2 Complete BPMN ontology

The proposed optional BPM module considered a set of minimum classes (to represent activities, processes and cases) and properties (connecting them together and linking them to log entries). In order to extend the metadata associated with business processes (e.g. the concrete flow of activities, potential parallel tasks, disjoint tasks, etc.) we propose to use the Business Process Model and Notation (BPMN) ontology[4] [13]. This BPMN ontology formalizes the well-

[3] `http://onprom.inf.unibz.it/`

[4] `https://dkm.fbk.eu/bpmn-ontology`

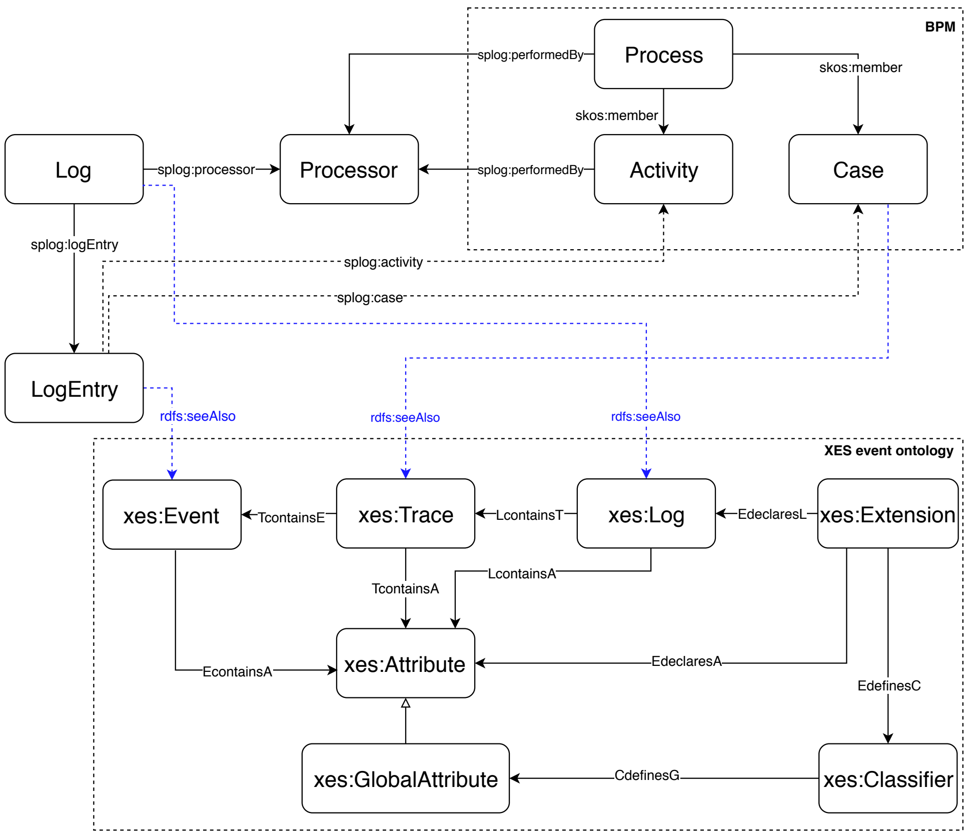

Figure 3.3: Relationships between the SPECIAL Policy Log vocabulary Extension and the XES event ontology.

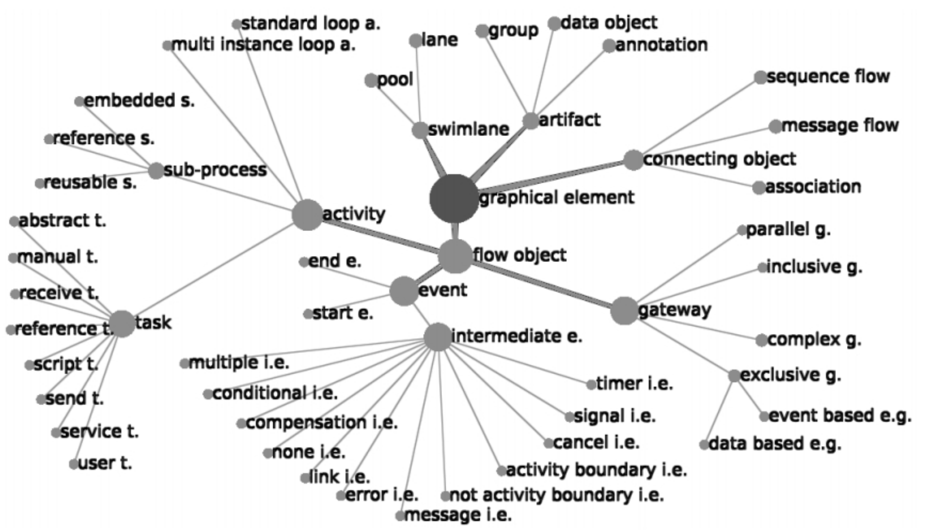

Figure 3.4: Overview of the taxonomy of the BPMN ontology (from [13]).

known Business Process Modelling Notation (BPMN) [17], one of the state-of-the-art graphical languages for BPMN.

Figure 3.4 depicts the taxonomy of the BPMN ontology, which currently contains 187 classes and 1447 axioms (v 0.4). Thus, BPMN-focus scenarios can make intensive use of existing BPMN design tools, which can potentially export models as instances of the BPMN ontology. Such instances can then be linked in our SPECIAL log using the provided classes and properties or extending them to capture more fine-grained information.

Appendix

The SPECIAL Policy Log Vocabulary

```
@prefix : <http://www.specialprivacy.eu/langs/splog#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix dcat: <http://www.w3.org/ns/dcat#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix spl: <http://www.specialprivacy.eu/langs/usage-policy#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<http://www.specialprivacy.eu/langs/splog> a owl:Ontology ;
rdfs:seeAlso "https://aic.ai.wu.ac.at/qadlod/policyLog/" ;
owl:versionInfo "0.5"@en .
#
#
# #################################################################
# #
# #    Object Properties
# #
# #################################################################
#
#
# http://www.specialprivacy.eu/langs/splog#RDFData

:RDFData a owl:ObjectProperty ;
rdfs:domain :DataDistribution .
#
# http://www.specialprivacy.eu/langs/splog#activity

:activity a owl:ObjectProperty ;
rdfs:domain :LogEntry ;
rdfs:range :Activity .
#
# http://www.specialprivacy.eu/langs/splog#case

:case a owl:ObjectProperty ;
rdfs:subPropertyOf owl:topObjectProperty ;
rdfs:domain :LogEntry ;
rdfs:range :Case .
#
# http://www.specialprivacy.eu/langs/splog#controller

:controller a owl:ObjectProperty ;
rdfs:subPropertyOf prov:agent ;
rdfs:domain :ConsentAssertion ;
rdfs:range :Controller .
#
# http://www.specialprivacy.eu/langs/splog#dataDistribution

:dataDistribution a owl:ObjectProperty ;
```

```
rdfs:domain :InstanceData ;
rdfs:range :DataDistribution .
#
# http://www.specialprivacy.eu/langs/splog#dataSubject

:dataSubject a owl:ObjectProperty ;
rdfs:subPropertyOf prov:wasAssociatedWith ;
rdfs:domain :LogEntry ;
rdfs:range :DataSubject .
#
# http://www.specialprivacy.eu/langs/splog#dataSubjectGroup

:dataSubjectGroup a owl:ObjectProperty ;
rdfs:subPropertyOf prov:wasAssociatedWith ;
rdfs:domain :DataSubjectGroup ;
rdfs:range :DataSubject .
#
# http://www.specialprivacy.eu/langs/splog#dimension

:dimension a owl:ObjectProperty ;
rdfs:subPropertyOf :logEntryContent ;
rdfs:domain :LogEntryGroup ;
rdfs:range :LogEntryContent .
#
# http://www.specialprivacy.eu/langs/splog#entryMember

:entryMember a owl:ObjectProperty ;
rdfs:subPropertyOf <http://www.w3.org/2004/02/skos/core#member> ;
rdfs:domain :LogEntryGroup ;
rdfs:range :LogEntry .
#
# http://www.specialprivacy.eu/langs/splog#hashAlgorithm

:hashAlgorithm a owl:ObjectProperty ;
rdfs:domain :ImmutableRecord ;
rdfs:range :HashAlgorithm .
#
# http://www.specialprivacy.eu/langs/splog#hashKeyLength

:hashKeyLength a owl:ObjectProperty ;
rdfs:subPropertyOf owl:topObjectProperty ;
rdfs:domain :ImmutableRecord ;
rdfs:range :HashKeyLength .
#
# http://www.specialprivacy.eu/langs/splog#immutableRecord

:immutableRecord a owl:ObjectProperty ;
rdfs:subPropertyOf prov:wasGeneratedBy ;
rdfs:domain :LogEntry ;
rdfs:range :ImmutableRecord .
#
# http://www.specialprivacy.eu/langs/splog#instanceData
```

```
:instanceData a owl:ObjectProperty ;
rdfs:domain :DataEvent ;
rdfs:range :InstanceData .
#
# http://www.specialprivacy.eu/langs/splog#logEntry

:logEntry a owl:ObjectProperty ;
rdfs:subPropertyOf prov:wasGeneratedBy ;
rdfs:domain :Log ;
rdfs:range :LogEntry .
#
# http://www.specialprivacy.eu/langs/splog#logEntryContent

:logEntryContent a owl:ObjectProperty ;
rdfs:domain :LogEntry ;
rdfs:range :LogEntryContent ;
rdfs:comment "Associates the Event with its content" .
#
# http://www.specialprivacy.eu/langs/splog#logEntryGroup

:logEntryGroup a owl:ObjectProperty ;
rdfs:subPropertyOf prov:wasGeneratedBy ;
rdfs:domain :Log ;
rdfs:range :LogEntryGroup .
#
# http://www.specialprivacy.eu/langs/splog#performedBy

:performedBy a owl:ObjectProperty ;
rdfs:domain :Activity ;
rdfs:range prov:Agent .
#
# http://www.specialprivacy.eu/langs/splog#processor

:processor a owl:ObjectProperty ;
rdfs:subPropertyOf prov:agent ;
rdfs:domain :Log ;
rdfs:range :Processor .
#
# http://www.specialprivacy.eu/langs/splog#recipient

:recipient a owl:ObjectProperty ;
rdfs:domain :SharingEvent ;
rdfs:range :Recipient .
#
# http://www.specialprivacy.eu/langs/splog#revoke

:revoke a owl:ObjectProperty ;
rdfs:domain :ConsentRevocation ;
rdfs:range :ConsentAssertion .
#
# http://www.specialprivacy.eu/langs/splog#subjectMember
```

```
:subjectMember a owl:ObjectProperty ;
rdfs:subPropertyOf <http://www.w3.org/2004/02/skos/core#member> .
#
# http://www.w3.org/2004/02/skos/core#member

<http://www.w3.org/2004/02/skos/core#member> a owl:ObjectProperty .
#
# http://www.w3.org/ns/prov#agent

prov:agent a owl:ObjectProperty .
#
# http://www.w3.org/ns/prov#wasAssociatedWith

prov:wasAssociatedWith a owl:ObjectProperty .
#
# http://www.w3.org/ns/prov#wasGeneratedBy

prov:wasGeneratedBy a owl:ObjectProperty .
#
#
#
# #################################################################
# #
# #    Data properties
# #
# #################################################################
#
#
# http://purl.org/dc/terms/issued

dct:issued a owl:DatatypeProperty .
#
# http://www.specialprivacy.eu/langs/splog#contentHash

:contentHash a owl:DatatypeProperty ;
rdfs:domain :ImmutableRecord ;
rdfs:range xsd:base64Binary .
#
# http://www.specialprivacy.eu/langs/splog#message

:message a owl:DatatypeProperty ;
rdfs:domain :LogEntry ;
rdfs:range xsd:string .
#
# http://www.specialprivacy.eu/langs/splog#rawData

:rawData a owl:DatatypeProperty ;
rdfs:domain :DataDistribution ;
rdfs:range xsd:string .
#
# http://www.specialprivacy.eu/langs/splog#transactionTime
```

```
:transactionTime a owl:DatatypeProperty ;
rdfs:subPropertyOf dct:issued ;
rdfs:domain :LogEntry ;
rdfs:range xsd:dateTimeStamp .
#
# http://www.specialprivacy.eu/langs/splog#userHash

:userHash a owl:DatatypeProperty ;
rdfs:domain :ImmutableRecord ;
rdfs:range xsd:base64Binary .
#
# http://www.specialprivacy.eu/langs/splog#validityEndTime

:validityEndTime a owl:DatatypeProperty ;
rdfs:domain :LogEntryGroup ;
rdfs:range xsd:dateTimeStamp .
#
# http://www.specialprivacy.eu/langs/splog#validityStartTime

:validityStartTime a owl:DatatypeProperty ;
rdfs:domain :LogEntryGroup ;
rdfs:range xsd:dateTimeStamp .
#
# http://www.specialprivacy.eu/langs/splog#validityTime

:validityTime a owl:DatatypeProperty ;
rdfs:subPropertyOf prov:atTime ;
rdfs:domain :LogEntry ;
rdfs:range xsd:dateTimeStamp .
#
# http://www.w3.org/ns/prov#atTime

prov:atTime a owl:DatatypeProperty .
#
#
#
# #################################################################
# #
# #    Classes
# #
# #################################################################
#
#
# http://www.specialprivacy.eu/langs/splog#Activity

:Activity a owl:Class ;
rdfs:comment "a BPM activity"@en ;
rdfs:label "Activity"@en .
#
# http://www.specialprivacy.eu/langs/splog#Case
```

```
:Case a owl:Class ;
rdfs:comment "a BPM case"@en ;
rdfs:label "Case"@en .
#
# http://www.specialprivacy.eu/langs/splog#ConsentAssertion

:ConsentAssertion a owl:Class ;
rdfs:subClassOf :PolicyEntry ;
rdfs:comment "A consent provided by a data subject"@en ;
rdfs:label "Consent Assertion"@en .
#
# http://www.specialprivacy.eu/langs/splog#ConsentRevocation

:ConsentRevocation a owl:Class ;
rdfs:subClassOf :PolicyEntry ;
rdfs:comment "The revocation of a given consent"@en ;
rdfs:label "Consent Revocation"@en .
#
# http://www.specialprivacy.eu/langs/splog#Controller

:Controller a owl:Class ;
rdfs:subClassOf prov:Agent ;
rdfs:comment "Controller as defined by Art. 4 (7) of the GDPR"@en ;
rdfs:label "Controller"@en .
#
# http://www.specialprivacy.eu/langs/splog#DataDistribution

:DataDistribution a owl:Class ;
rdfs:subClassOf dcat:Distribution .
#
# http://www.specialprivacy.eu/langs/splog#DataEvent

:DataEvent a owl:Class ;
rdfs:subClassOf :LogEntry ;
rdfs:comment "Log entries that are actually events on the data, such
   as data processing and sharing events"@en ;
rdfs:label "Data Event"@en .
#
# http://www.specialprivacy.eu/langs/splog#DataSubject

:DataSubject a owl:Class ;
rdfs:subClassOf prov:Agent ;
rdfs:comment "Natural person as per Art. 4 (1) of the GDPR"@en ;
rdfs:label "Data Subject"@en .
#
# http://www.specialprivacy.eu/langs/splog#DataSubjectGroup

:DataSubjectGroup a owl:Class ;
rdfs:subClassOf :DataSubject ;
rdfs:comment "A goup of one or more data subjects"@en ;
rdfs:label "Data Subject Group"@en .
#
```

```
# http://www.specialprivacy.eu/langs/splog#HashAlgorithm

:HashAlgorithm a owl:Class ;
rdfs:comment "Defines an algorithm for hashing"@en ;
rdfs:label "Hash Algorithm"@en .
#
# http://www.specialprivacy.eu/langs/splog#HashKeyLength

:HashKeyLength a owl:Class ;
rdfs:comment "Defines the key length of a Hash Algorithm"@en ;
rdfs:label "Hash Key Length"@en .
#
# http://www.specialprivacy.eu/langs/splog#ImmutableRecord

:ImmutableRecord a owl:Class ;
rdfs:subClassOf prov:Activity ;
rdfs:comment "the immutable record of an event"@en ;
rdfs:label "Immutable Record"@en .
#
# http://www.specialprivacy.eu/langs/splog#InstanceData

:InstanceData a owl:Class ;
rdfs:subClassOf dcat:Dataset .
#
# http://www.specialprivacy.eu/langs/splog#Log

:Log a owl:Class ;
rdfs:subClassOf prov:Entity ;
rdfs:comment "A Log is a collection of data that records data
    processing and sharing events as well as consent-related
    activities (acquisition and revocation)"@en ;
rdfs:label "Log"@en .
#
# http://www.specialprivacy.eu/langs/splog#LogEntry

:LogEntry a owl:Class ;
rdfs:subClassOf prov:Activity ;
rdfs:comment "A log entry contains information about a processing and
sharing  event associated to a data subject, as well as actions
related  to the consent provided (or revoked) by a data subject"@en ;
rdfs:label "Log Entry"@en .
#
# http://www.specialprivacy.eu/langs/splog#LogEntryContent

:LogEntryContent a owl:Class ;
rdfs:subClassOf spl:Authorization ;
rdfs:comment "The content of a log entry in terms of the data involved,
how is data processed, the purpose of the process, where and for how
long is the data stored and potential disclosures to third parties"@en ;
rdfs:label "Log Entry Content"@en .
#
# http://www.specialprivacy.eu/langs/splog#LogEntryGroup
```

```
:LogEntryGroup a owl:Class ;
rdfs:subClassOf :LogEntry ;
rdfs:comment "a log entry group contains information about one or more
log  entries"@en ;
rdfs:label "Log Entry Group"@en .
#
# http://www.specialprivacy.eu/langs/splog#PolicyEntry

:PolicyEntry a owl:Class ;
rdfs:subClassOf :LogEntry ;
rdfs:comment "Log entries related to policies and consent"@en ;
rdfs:label "Policy Entry"@en .
#
# http://www.specialprivacy.eu/langs/splog#Process

:Process a owl:Class ;
rdfs:comment "a BPM process"@en ;
rdfs:label "Process"@en .
#
# http://www.specialprivacy.eu/langs/splog#ProcessingEvent

:ProcessingEvent a owl:Class ;
rdfs:subClassOf :DataEvent ;
rdfs:comment "A data processing event"@en ;
rdfs:label "Data Processing Event"@en .
#
# http://www.specialprivacy.eu/langs/splog#Processor

:Processor a owl:Class ;
rdfs:subClassOf prov:Agent ;
rdfs:comment "Processor as defined by Art. 4 (8) of the GDPR"@en ;
rdfs:label "Processor"@en .
#
# http://www.specialprivacy.eu/langs/splog#Recipient

:Recipient a owl:Class ;
rdfs:subClassOf prov:Agent ;
rdfs:comment "recipient as defined by Art. 4 (9) of the GDPR"@en ;
rdfs:label "Recipient"@en .
#
# http://www.specialprivacy.eu/langs/splog#SharingEvent

:SharingEvent a owl:Class ;
rdfs:subClassOf :DataEvent ;
rdfs:comment "A data sharing event"@en ;
rdfs:label "Data Sharing Event"@en .
#
# http://www.specialprivacy.eu/langs/usage-policy#Authorization

spl:Authorization a owl:Class .
#
```

```
# http://www.w3.org/ns/prov#Activity

prov:Activity a owl:Class .
#
# http://www.w3.org/ns/prov#Agent

prov:Agent a owl:Class .
#
# http://www.w3.org/ns/prov#Entity

prov:Entity a owl:Class .
```

Bibliography

[1] S. Bradner. Key words for use in rfcs to indicate requirement levels, 1997. URL `https://tools.ietf.org/html/rfc2119`.

[2] D. Calvanese, M. Montali, A. Syamsiyah, and W. M. van der Aalst. Ontology-driven extraction of event logs from relational databases. In *International Conference on Business Process Management*, pages 140–153. Springer, 2015.

[3] G. Carothers and A. Seaborne. Rdf 1.1 turtle: Rdf dataset language. *W3C Recommendation, February*, 2014.

[4] Dublin Core metadata initiative. Dcmi metadata terms, 2012. URL `http://dublincore.org/documents/dcmi-terms/`.

[5] S. H. Garlik, A. Seaborne, and E. Prud'hommeaux. SPARQL 1.1 Query Language, W3C Recommendation, 2013. URL `https://www.w3.org/TR/sparql11-query/`.

[6] S. Gerbracht. Possibilities to encrypt an rdf-graph. In *Information and Communication Technologies: From Theory to Applications*, pages 1–6. IEEE, 2008.

[7] M. Giereth. On partial encryption of rdf-graphs. In *International Semantic Web Conference*, pages 308–322. Springer, 2005.

[8] C. W. Günther and E. Verbeek. Xes standard definition. *Fluxicon Process Laboratories*, 13:14, 2009.

[9] T. Lebo, S. Sahoo, and D. McGuinness. Prov-o: The prov ontology. *W3C Recommendation, April*, 2013.

[10] B. F. Lóscio, C. Burle, and N. Calegari. Data on the web best practices. *W3C Recommendation 31 January 2017*, 2017.

[11] L. T. Ly, F. M. Maggi, M. Montali, S. Rinderle-Ma, and W. M. van der Aalst. Compliance monitoring in business processes: Functionalities, application, and tool-support. *Information systems*, 54:209–234, 2015.

[12] E. Prud'hommeaux and G. Carothers. Rdf 1.1 turtle: Terse rdf triple language. *W3C Recommendation, February*, 2014.

[13] M. Rospocher, C. Ghidini, and L. Serafini. An ontology for the Business Process Modelling Notation. In *Proc. of Formal Ontology in Information Systems*, volume 267, pages 133–146. IOS Press, 2014.

[14] H. Van de Sompel, M. Nelson, and R. Sanderson. Http framework for time-based access to resource states–memento. Technical report, IETF, 2013. RFC7089, https://tools.ietf.org/html/rfc7089.

[15] W. Van Der Aalst, A. Adriansyah, A. K. A. De Medeiros, F. Arcieri, T. Baier, T. Blickle, J. C. Bose, P. Van Den Brand, R. Brandtjen, J. Buijs, et al. Process mining manifesto. In *International Conference on Business Process Management*, pages 169–194. Springer, 2011.

[16] W. M. Van der Aalst. Process mining. In *Process Mining*, pages 95–123. Springer, 2011.

[17] S. A. White et al. Business process modeling notation. *Specification, BPMI. org*, page 21, 2004.