
# Initial specification and prototyping of the process re-engineering framework

## Deliverable D4.1

**Editor**

Azadeh Mozafari Mehr (TUE)
Mariza Koukovini (ABOVO)

**Reviewers**

Nikolaos Dellas (SLG)

**Date**

27th September 2019

**Classification**

Public

BPR4GDPR has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 787149

## Contributing Author(s)

| Name | Partner |
|---|---|
| Azadeh Mozafari Mehr | TUE |
| Mariza Koukovini | ABOVO |
| Azadeh Mozafari Mehr, Mariza Koukovini | TUE, ABOVO |
| Azadeh Mozafari Mehr, Mariza Koukovini | TUE, ABOVO |

## # Version History

| # | Description |
|---|---|
| 1 | 19/09/2019 — Structure, template and TUE contribution |
| 2 | 25/09/2019 — First integrated version of TUE and ABOVO contributions |
| 3 | 26/09/2019 — Second integrated version of TUE and ABOVO contributions |
| 4 | 27/09/2019 — Integrating the review of SLG |

## Table of Contents

# 1 Introduction

This document is prepared in the context of Work Package 4 “Privacy-aware process re-engineering”, and constitutes its first deliverable. In line with the objectives of WP4, the present deliverable D4.1 aims to address the initial specification and prototyping of the process re-engineering framework (Tasks T4.1, T4.2, T4.3), taking into account the Initial Specification of BPR4GDPR architecture (D2.3) and Use cases and requirements (D2.1). As illustrated in Figure 1, the work described in this deliverable is related to four of the eight BPR4GDPR operational phases.

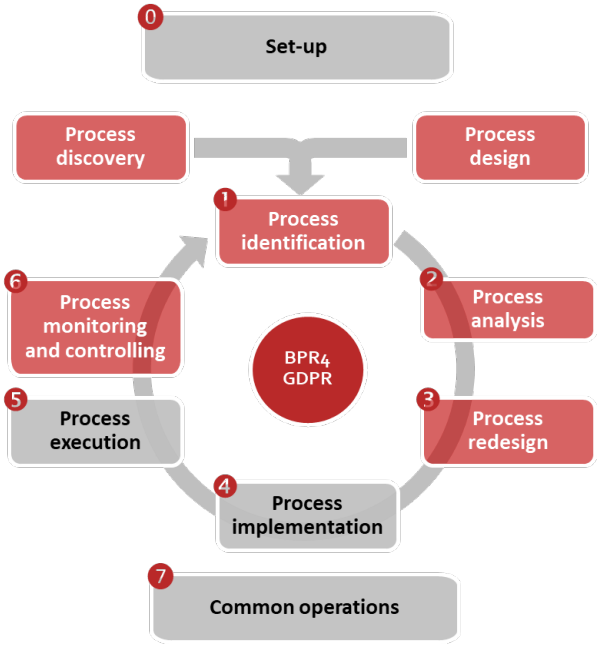


**Figure 1: BPR4GDPR operational phases related to this deliverable**

**Process identification** concerns the definition of process models, leveraging two possible ways: i) process discovery mechanisms, in order to depict, in a formal way, the procedures and associated information flows taking place within an organisation, through the resulting process models; ii) administrative users defining such procedures themselves using the appropriate graphical tool. **Process analysis** concerns the analysis of a process model in order to identify the risks, flaws and points of non-compliance, on the basis of well-defined policies. This way, process models shall be evaluated and verified as regards their compliance with the GDPR and provisions thereof. **Process redesign** complements process analysis, by providing for the automatic transformation of non-compliant process models, so that they are rendered inherently privacy-aware before being deployed for execution. Finally, **Process monitoring and controlling** concerns the use of process mining for the ex post analysis of processes, in order to ensure that specified policies are indeed enforced, fostering accountability.

From an architectural point of view, this deliverable is associated with two of the main architectural domains identified in the BPR4GDPR architecture, notably *Planning* and *Monitoring* (Figure 2). The former concerns the specification of process models, their verification as regards compliance with the GDPR and their subsequent transformation, if needed, so that they become compliant *by design*. The latter deals with process mining and monitoring with the aim to identify discrepancies between compliant processes and actual process behaviour.

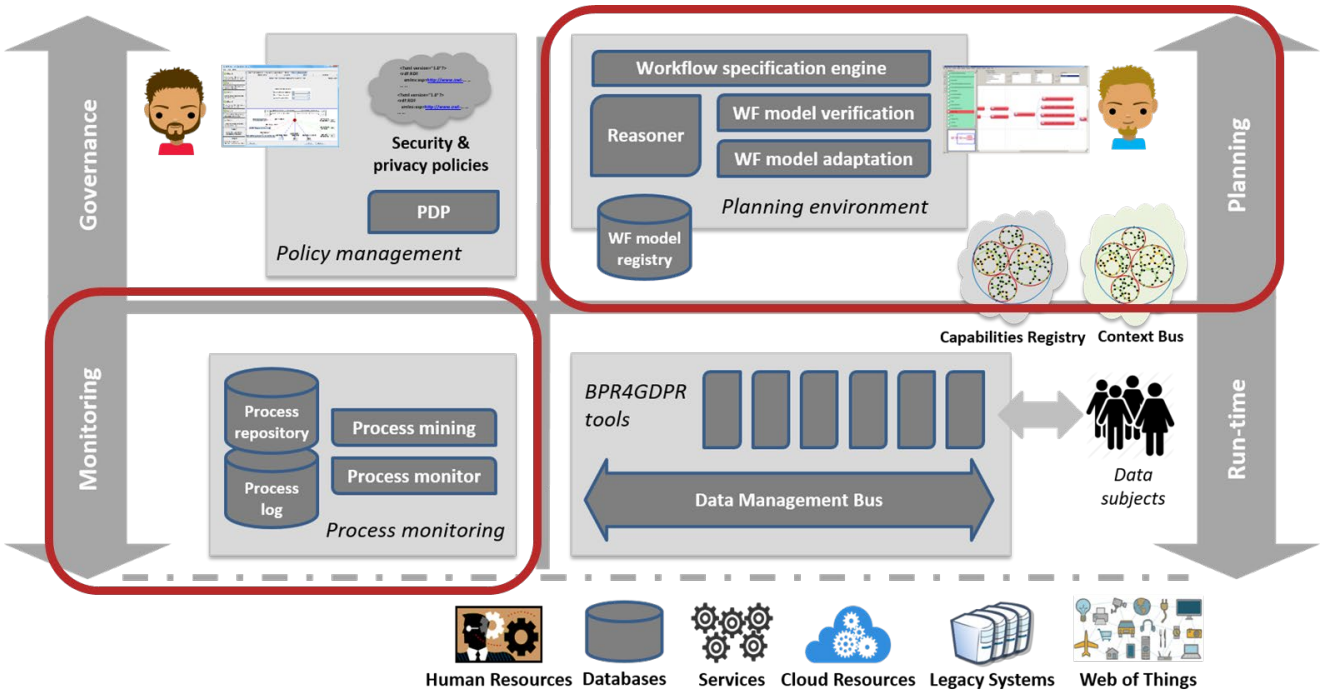


**Figure 2: D4.1 in relation with the BPR4GDPR architecture**

The specification of this deliverable includes the high-level functional structure of the process re-engineering framework and its constituent tools; it will ultimately evolve into the final specification and prototyping of the process re-engineering framework (D4.2). Its structure is as follows.

Section 2 deals with the initial stages of process lifecycle, notably process identification by means of design, analysis and redesign. It provides the description of the BPR4GDPR Compliance Metamodel, along with the mechanisms for process analysis and re-engineering.

Section 3 explains business process re-engineering stages based on applying different process mining techniques such as automated process discovery and compliance/conformance checking. This section shows how the outputs of different types of process mining helps fulfilling BPR tasks.

Section 4 details the specification and architecture of the process discovery and mining tool of the process re-engineering framework. The prototype tool is based on the Right to be Forgotten provision of the GDPR. Process discovery and compliance checking mechanisms of the tool are detailed through accompanied screen shots, whereas the tool and sample event log(s) are provided online.

Section 5 provides an overview and envisaged scheme of the specifications of the final prototype of the re-engineering framework (D4.2). The final prototype will provide a process management tool entailing process verification and adaptation, as well as automated process generation and adaptability functionalities.

Section 6 concludes this deliverable with providing an overview of the initial process re-engineering framework and some future targets.

## 2 Process Planning

Business processes are characterised by serious privacy implications, since they natively rely to a large extent on access to and exchange of data. In fact, privacy pertains to all three core workflow perspectives, as it is closely related to the tasks being executed themselves (control perspective), the flow and processing of information (data perspective) and legitimate resource allocation (resource perspective). Besides, workflows are often based on and foster collaboration within heterogeneous environments and among many stakeholders, something that significantly complicates the direct and effective use of already existing solutions to privacy protection. The key challenge arising in such context is that the various activities must no longer be considered only "in isolation" but also with respect to operational and data flows, resulting in a holistic view across the corresponding procedures; in other words, required mechanisms (e.g., access control) must be effectively enforced regarding not only individual actions but also large-scale interrelations thereof at the workflow level. At the same time, *Privacy by design,* referring to the philosophy and approach of embedding privacy directly into the design and operating specifications of information technologies and systems [1], has long been recognised as an essential component of effective privacy protection.

Considering the above, BPR4GDPR proposes, as a first stepping stone towards GDPR-compliant processes, a design solution characterised by the following features: the inclusion, at the workflow model level, of structures able to support the *in-design* specification of privacy policies, leading to targeted privacy configurations enforceable at run-time; the automatic verification of workflow models against privacy provisions; their automatic transformation.

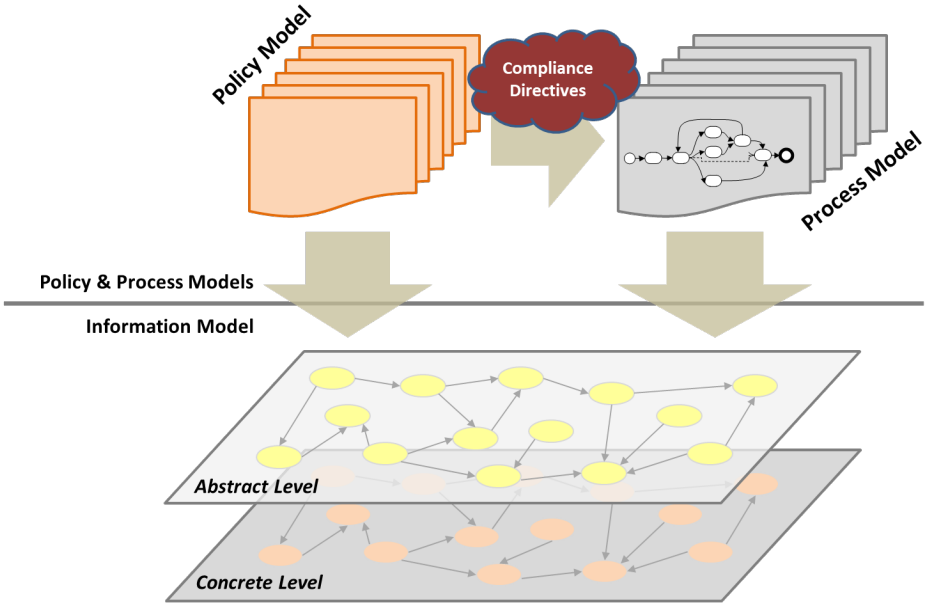


**Figure 3: BPR4GDPR foundational models**

Operationally, the general idea is the following: a workflow model, as specified by its designer, is subject to automatic verification and transformation in order to become privacy-aware. In this context, a number of models is leveraged, visualised in Figure 3. The *Compliance Metamodel* comprises an innovative ontological approach to workflow modelling, encompassing a variety of features for the incorporation of privacy aspects in workflow specifications. Its verification and transformation are based on access and usage control rules provided by a *Policy Model*, with the latter comprising a sophisticated approach for privacy-aware access control.

Reasoning in the Policy Model results in the formation of *Compliance Directives*, used to regulate the verification and transformation procedure. All the above are grounded on a semantically rich Information Model, expressed through the *Compliance Ontology* (cf. Section 2.1) and organised in two levels of representation, notably abstract and concrete. Its basic advantage is that it provides for fine-grained description of the underlying concepts and their relationships. The Compliance Ontology and the Policy model, as well as the Compliance Metamodel and Directives, are all implemented as ontologies, providing for high expressiveness, formal and machine-interpretable semantics, semantic consistency and interoperability, and inference of knowledge. The former two have been extensively documented in deliverables D3.1 and D3.2, respectively, while the current deliverable, after a short overview of the Compliance Ontology for reasons of completeness, will cover the Compliance Metamodel and the associated verification methodology.

In the sections that follow, the hypothetical workflow presented in Figure 4 will serve to clarify related concepts, inspired by the ePrescription scenario of Deliverable D2.1 “Use cases and requirements”. According to the depicted process, supposedly defined by a process administrator of an organisation in charge of providing ePrescription services such as BPR4GDPR partner IDIKA, through the modelling tool of the BPR4GDPR Planning Environment, an authorised doctor may search among the prescriptions of a patient and choose to edit or copy one of them or even create a new one; in any case, this is expected to result in the issuance of a prescription, to be subsequently dispensed by a pharmacist. The mission of the Planning Environment is to ensure that this process is specified in a GDPR-compliant manner by checking the process model and complementing/modifying it as necessary, based on the tools and procedures presented herein.

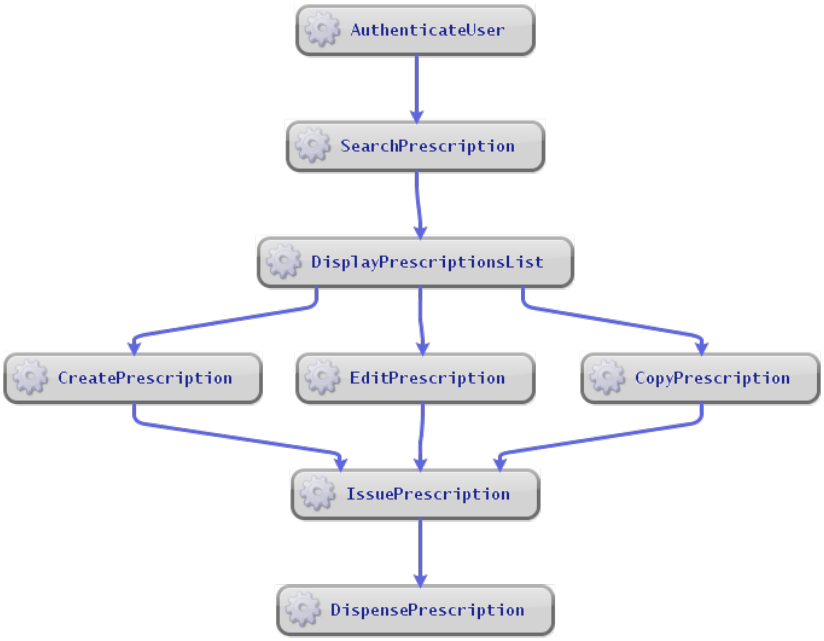


**Figure 4: Example workflow model**

## 2.1 Compliance ontology

The Compliance Ontology documented in detail in the Deliverable D3.1 “Compliance ontology specification” is a generic and, at the same time, highly expressive model ultimately grounded on the analysis of the GDPR that could be easily mapped to the underlying domain-specific information model of any organisation; it actually provides a high-level codification of the GDPR, by extracting the concepts that need to be addressed by the

BPR4GDPR policy framework, as well as by the privacy-aware process reengineering. In that respect, it constitutes the Information Model upon which the policy framework is established.

### 2.1.1 Overview of the Information Model

The day-to-day operation of an organisation involves a variety of entities, like machines, users and data. BPR4GDPR considers two representation levels; the *concrete level* refers to well-specified entities, e.g., named humans, while the *abstract level* enables referring to entities by using abstractions, especially their semantic type and attributes. The main entities of the two levels are summarised in Table 1.

More specifically, at a concrete level, the set of *Users (U)* represents human entities, while this of *Organisations (Org)* describes internal divisions (e.g., departments) or external parties (e.g., sub-contractors). The machinery comprises the *Machines (M)* set, providing hosting to *Operation Containers (OpC)* that offer *Operation Instances (OpI)*. The latter correspond to actual implementations of functionalities, while Operation Containers bundle collections of Operation Instances provided by the same functional unit. Finally, information comprises the set of *Data (D)*, whereas *Events (E)* take place and may lead to actions for responding thereto.

| Abstract Level | Concrete Level | Description |
|---|---|---|
| Data Types (*DT*) | Data (*D*) | Data being collected and/or processed, organised according to their semantic types |
| Roles (*R*) | Users (*U*) | Human users assigned with roles reflecting their responsibilities inside an organisation |
| Operations (*Op*) | Operation Instances (*OpI*) | Operations reflect all actions that can take place in the context of the system's operation |
| Operation Container Types (*OpCT*) | Operation Containers (*OpC*) | Components or other functional structures that typically offer a set of operations together |
| Machine Types (*MT*) | Machines (*M*) | Hardware (in the typical case) components hosting operation containers |
| Organisation Types (*OrgT*) | Organisations (*Org*) | The various domains within which actions are performed |
| Event Types (*ET*) | Events (*E*) | Expected or unexpected events that may affect the operation of an organisation, or may call for actions in response |
| Context Types (*ConT*) | Context keys and values | Real-time parameters that should be considered in decision making, such as spatial, temporal, environmental values |
| Purposes (*Pu*) | (no concrete representation) | Purposes for which actions take place, processes are executed, and access to resources is requested |
| Attributes (*Att*) | Attribute keys and values | Characteristics further describing members of the other sets |

**Table 1: Concepts of the Information Model**

All above elements constitute instantiations of their semantic equivalents described at the abstract level. Users are assigned with *Roles (R)*, Operation Instances provide implementations of *Operations (Op)*, while data, organisations, machines, operation containers, and events have types, reflecting the semantic class they fall under; thus, sets of *Data Types (DT)*, *Organisation Types (OrgT)*, *Machine Types (MT)*, *Operation Container Types (OpCT)* and *Event Types (ET)* are defined. The semantic model also includes *Context Types (ConT)*, enabling the definition of contextual parameters, *Attributes (Att)*, leveraged for describing properties and characteristics of other elements, and *Purposes (Pu)* justifying access requests, as well as any other type of action that takes place during the system operation.

All concepts summarised in Table 1 comprise graphs of elements that are characterised by relations; the latter are implemented by predicates defining AND- and OR-hierarchies and enabling the inheritance of attributes and rules, as well as the specification of dependencies. For instance, and with respect to the *DT* graph, three partial order relations are defined: $isA(dt_i, dt_j)$, $moreDetailedThan(dt_i, dt_j)$ and $isPartOf(dt_i, dt_j)$, where $dt_i, dt_j \in DT$, reflecting the particularisation of a concept, the detail level and the inclusion of some data types to another, respectively. Figure 5 provides a simple example of the *DT* graph hierarchies, highlighting all three relations.

Moreover, the model specifies the necessary predicates in order to link concepts from different graphs; for example, the predicate $mayActForPurposes(r, \langle pu \rangle^k)$, where $r \in R$, $\langle pu \rangle^k \subseteq \mathcal{P}(Pu)$, indicates the legitimate purposes $\langle pu \rangle^k$ for which the users assigned with the role $r$ may act.

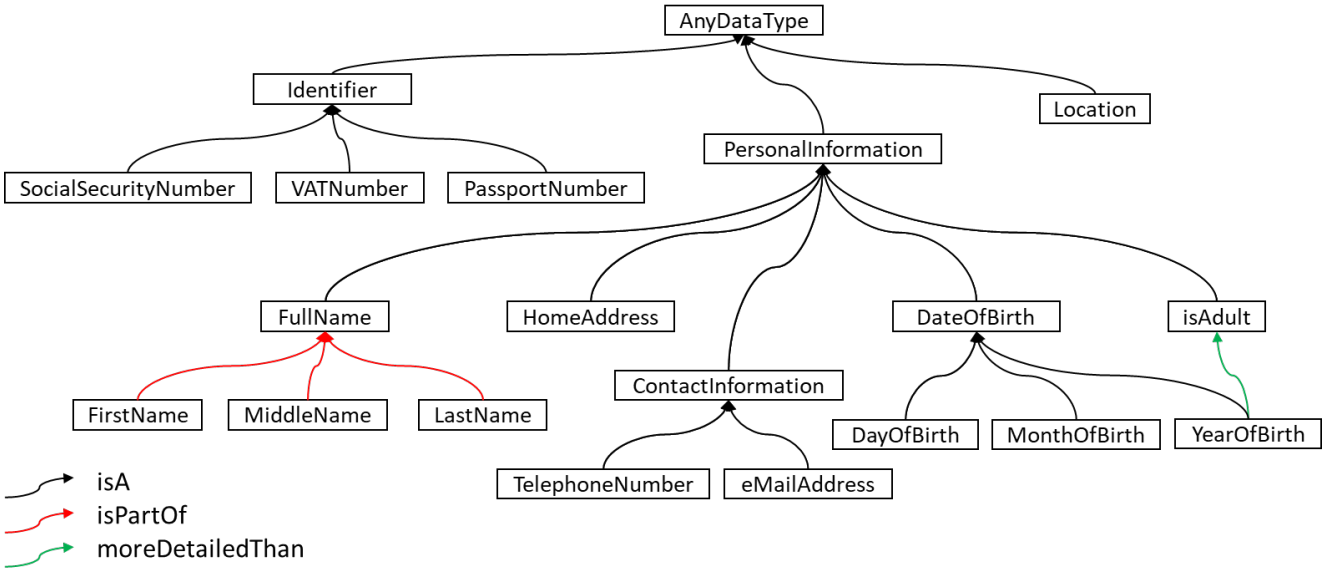


**Figure 5: Information Model hierarchy example**

### 2.1.2 Information model ontology

Figure 6 illustrates the Information Model Ontology (IMO), providing the ontological implementation of the Information Model. All abstract concepts described in the previous Section comprise classes, whereas their intra- and inter-class relations are implemented as OWL object properties. In this context, the main classes comprising the BPR4GDPR Information Model Ontology are the following:

- `DataTypes`, which comprises all the types of data that can be used during the system's operation.
- `Roles`, which includes the roles that the system's users may hold.
- `Operations`, which contains all the operations that may take place.
- `OrganisationTypes`, reflecting the different types of organisations, external or internal, actual or virtual, that may be involved in operations.
- `OperationContainerTypes`, representing components that offer a set of operations together.
- `MachineTypes`, containing the types of machines that are used during the system's operation.
- `EventTypes`, reflecting the types of events that may affect the operations and/or may require some action to take place in response.
- `ContextTypes`, serving for the specification of run-time constraints.
- `Attributes`, providing for the description of properties that characterise the entities.

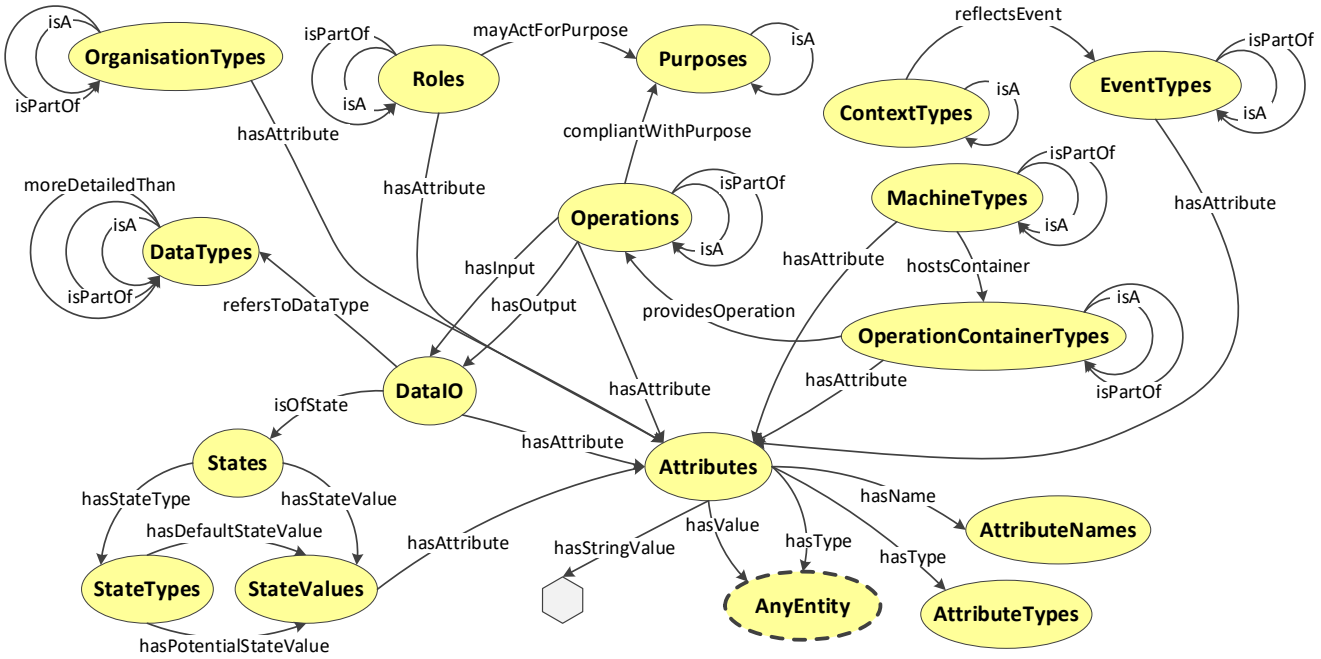


**Figure 6: Information Model Ontology**

These classes are complemented by additional classes holding a rather "auxiliary" role; these enable, for instance, to define the inputs and output of an operation, the state of data as regards some of their properties (e.g., anonymised or not), etc.

On the other hand, the main intra-class properties are `isA`, `isPartOf` and `moreDetailedThan` that, along with their inverses[1], essentially comprise AND- and OR- hierarchies, enabling inheritance, as well as dependencies specification. Associations between concepts of different classes are implemented by means of inter-class OWL object properties; for instance, the roles that may act for a purpose are indicated by the `mayActForPurpose` property, whereas the attributes characterising a concept are related to the concept by means of the `hasAttribute` property.

## 2.2 Compliance metamodel

Processes are typically implemented through workflows, and thus these two terms will henceforth be used interchangeably. In general terms, a workflow describes a series of actions with well-defined sequential relations and information dependencies among them. A workflow under execution is referred to as a *workflow instance*, whereas its specification is provided by a *workflow model*. The BPR4GDPR modelling approach has been designed with the aim to inherently support the prescription of provisions necessary for GDPR compliance and as such comprises the *compliance metamodel*, implemented as an ontology.

The most fundamental artefacts of a workflow model are *tasks* and *flows*. The former represent actions to be executed within the workflow, each describing the *operation* performed by an *actor* on an *asset*. Flows express dependencies between tasks, are represented through directed edges and are of two types: *control* and *data*.

[1] Inverse properties are explicitly defined for all object properties in the ontology, in order to ease navigation from one ontological element to another.

A control flow dependency $t_A \xrightarrow{f_C} t_B$ between two tasks $t_A$ and $t_B$ means that $t_B$ is executed only after the execution of $t_A$ is completed; what the edge transfers is the thread of control, potentially accompanied by the necessary control parameters. On the contrary, a data flow dependency $t_A \xrightarrow{f_d} t_B$ assumes that actual data of interest are exchanged (i.e., to be accessed and processed by the destination task), denoting explicit data dependencies. Further, a workflow model is complemented by the operational *purposes* it is meant to serve, and the potential *initiators*, denoting entities authorised to initiate the workflow. Therefore, a *workflow model* can be defined as a tuple $\langle T, F_C, F_D, Init, WFPu \rangle$, such that: $T$ is a finite set of tasks $\langle t_1, t_2, \dots, t_n \rangle$; $F_C$ and $F_D$ are sets of directed edges, expressing the control flow and data flow relations among tasks; $Init$ is the set of human actors that, according to the given specification, are allowed to trigger the workflow execution; $WFPu \sqsubseteq Pu$ denotes the set of purposes for which the workflow is intended to be executed.

The core part of the Workflow Model Ontology (WMO) is shown in Figure 7. All concepts participating in a workflow specification are represented by instances of the corresponding classes, while OWL object properties model their relationships with each other and with IMO elements. Every workflow model is represented as an instance of the `WorkflowModels` class, comprising its reference semantic entry; this is associated to sets of `Initiators` and `WFPurposes` individuals, through the `initiatedBy` and `servesPurpose` properties. Class `Initiators` indicates, through property `refersToActor`, those `ActorEntities` (see below) constituting initiators of workflows, while `refersToPurpose` maps members of the class `WFPurposes` to IMO's `Purposes`.

Tasks and edges are represented through the classes `TaskNodes` and `Edges`, respectively, while the latter is further subclassed by `DataEdges` and `ControlEdges`, denoting the two types of flows, $F_C$ and $F_D$ . In that respect, the `includesTask` and `includesEdge` properties map a `WorkflowModels` instance with the tasks and edges contained therein. Additionally, given that flows constitute directed edges, each has exactly one source and one destination task; therefore, instances of `Edges` are associated with `TaskNodes` instances through the `hasSource` and `hasDestination` properties.

In Figure 8, the user-defined workflow depicted in 6 is being reproduced as an ontology[2]. For clarity reasons only `TaskNodes`, `ControlEdges` and `DataEdges` instances are included, along with their interrelations. The purposes the process is intended to serve are declared to be “PreventiveMedicine” and “MedicalDiagnosis”, while possible initiators are all persons holding either the role “HospitalDoctor” or the role “PrivateDoctor”.

[2] In the top-right section of Figure 6, the notation that will be used in the examples of the current Section is explained.

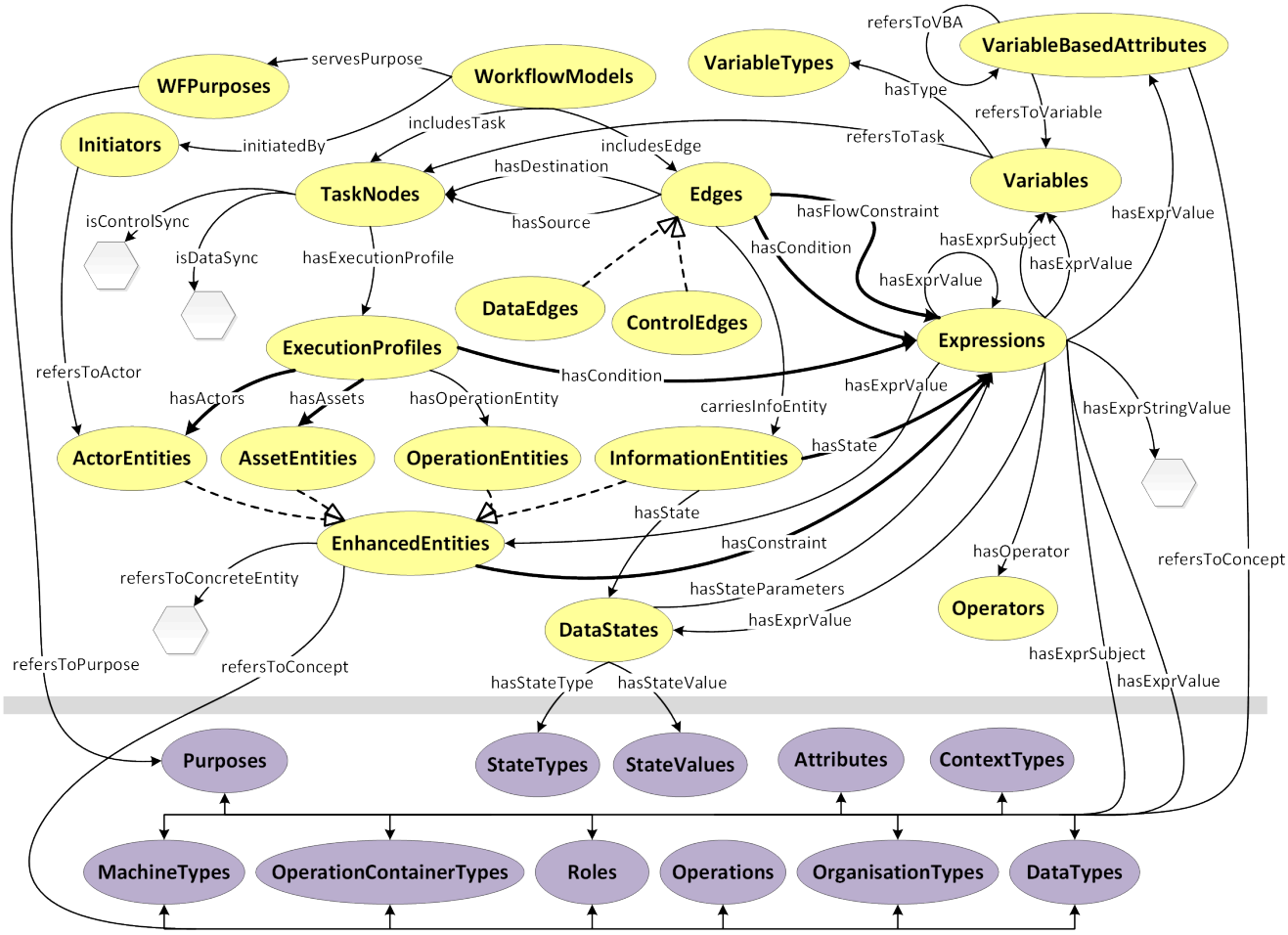


Figure 7: The Workflow Model Ontology (WMO). Dark-shaded ovals denote Compliance Ontology classes.

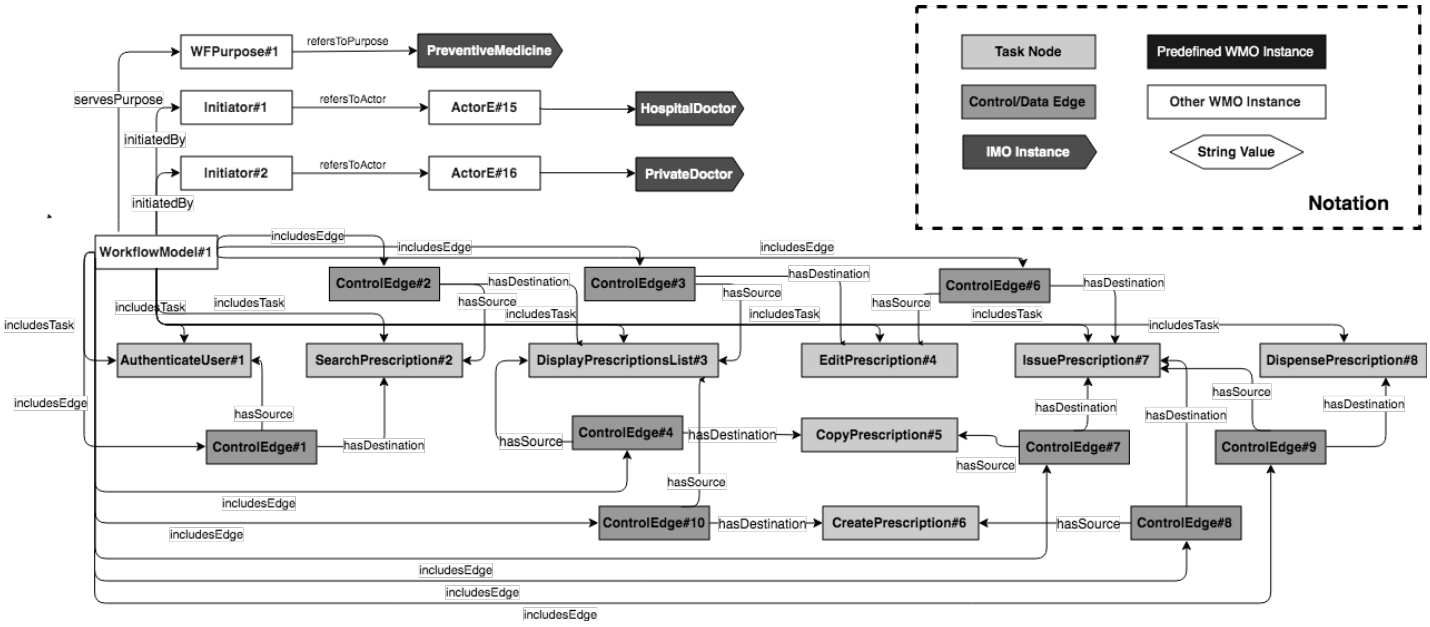


Figure 8: Ontological representation of the reference workflow.

Going a bit deeper, the core constituents of tasks are actors, operations and assets, while for flows, the exchanged information is essential for edges definition. In order to adequately capture the core workflow perspectives, a comprehensive approach for modelling these elements is adopted, centred around the notion of *enhanced entities*; the latter describe elements that their definition is either concrete, or abstract and constrained over attributes and/or subconcepts. In other words, an *enhanced entity* is defined as $\langle conConcept \mid abConcept, Constr \rangle$, where $conConcept$ denotes a concrete element; $abConcept$ denotes a concept expressed at the abstract level; and $Constr$ is an expression used to refer to concepts satisfying given conditions. Despite the semantic and structural differences of actors, assets, operations and information, the corresponding entities share common features and therefore, to some extent, a uniform representation; thus, the associated WMO classes are all subclasses of `EnhancedEntities`. Each instance indicates the entity's semantic type and the constraints that describe said abstract entity, through the `refersToConcept` and `hasConstraint` properties, while if the entity is defined concretely, the `refersToConcreteEntity` property is used instead.

Expressions in this context comprise ternary relations assigning a value to a subject through an operator, or logical structures of such triples. Specifically, an atomic expression is a tuple $\langle exprSubject, operator, exprValue \rangle$, such that: $exprSubject$ reflects the reference concept; $operator \in Operators$, the latter being a set of operators (e.g., *equals*, *greaterThan*, etc.); $exprValue$ represents the value assigned to the $exprSubject$. Logical structures thereof are in turn formalised through logical relations on atomic expressions.

Notably, logical relations are used in a broad context within the framework, in order to group various kinds of concepts towards achieving rich expressiveness. For instance, a task may not be assigned to one type of actor; its definition may include a set of heterogeneous entities that must jointly undertake its execution (AND), a set of alternative actors, inclusive (OR) or exclusive (XOR), or combinations thereof. Along this line, thick lines in Figure 7 imply the use of logical relations for structuring workflow elements; they are implemented by means of the `LogicalRelations` class (Figure 9). Instances of its subclasses `ANDRelations`, `ORRelations`, `XORRelations` represent the AND, OR and XOR operators. Ontological instances participating in logical relations are referenced through the `posRelatedTo` and `negRelatedTo` properties, with the latter modelling the use of the NOT operator. Expressions, on the other hand, are ontologically modelled by means of the `Expressions` and `LogicalRelations` classes; instances of the former essentially model atomic expressions, whereas the latter provide for structuring composite expressions. As shown, an expression's subject and object can be instances of the WMO or IMO, or arbitrary, i.e., concepts not defined semantically; in such case, the `hasExprStringValue` property is used instead of `hasExprValue`, for assigning a String value.

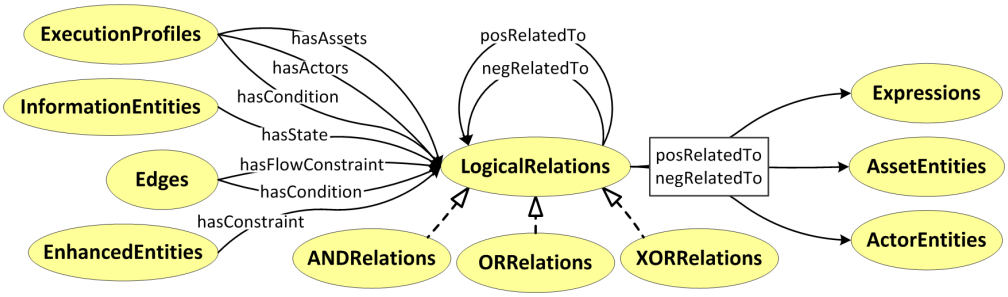


**Figure 9: Logical relations**

Moreover, in order to further enhance the expression of powerful constraints, *workflow variables* provide reference "handles" serving as abstractions of the underlying objects, thus allowing to describe relative relations and dependencies. For instance, a constraint such as "the actor of task $T_B$ should be the same as the one of $T_A$" can be defined, whatever the actors of $T_A$ and $T_B$ are. Workflow variables are grouped under the WMO `Variables` class. Further, a workflow variable has a type (a `VariablesTypes` instance, e.g., `Actor`, `Task`, etc.) and points to a task (`refersToTask` object property), when this is necessary, e.g., for denoting the actor of some task in the workflow. Each time a task is added, corresponding instances are created, so that the task's actor and asset, as concepts themselves, can be referenced by expressions. In the same direction, `VariableBasedAttributes` instances may be used for cases where the value of an expression refers to an attribute of a Variables instance, e.g., `Actor.Schedule`. Nested variable-based attributes are also supported.

Information entities in particular, used for flows specification, are further enriched with *states*, serving as indicators of the effect that the execution of preceding tasks has had on each information entity. A data state is characterised by a *type* and a *value*, incorporated in the Compliance Ontology as the sets $ST$ and $SV$, being the instances of IMO classes `StateTypes` and `StateValues`. A state may also be described through some parameters. Ontologically, each data state comprises an instance of the `DataStates` WMO class; its type, value and parameters are indicated by the `hasStateType`, `hasStateValue` and `hasStateParameters` properties (Figure 7), while the state itself is assigned to an `InformationEntity` by means of the `hasState` property. Nevertheless, when a certain state characterises parts of the referenced information entity and not the entity as a whole, `hasState` points to an appropriate expression. Therein, the `exprSubject` corresponds to the appropriate data type, the `exprValue` is a `DataStates` object, while the operator used is `inState`.

Based on the above described entities tasks and flows can subsequently be defined. The core element of a task is its *operation*, i.e., the functionality it implements. From there on, unlike other approaches where the definition of a task is "monolithic", this current framework introduces the concept of *execution profiles*, enabling the specification of variations regarding the execution of a task. This concerns two aspects: differentiated execution based on some *conditions*, and capturing the dependencies between the task's actors, assets and operation constraints, that is, precisely defining their valid combinations. Therefore, each task is associated with a non-empty set of execution profiles, each of which is a tuple $\langle \varphi(Actors_{WM}),\ \varphi(Assets_{WM}), oe, taskConditions \rangle$, such that: $\varphi(Actors_{WM})$, $\varphi(Assets_{WM})$ are logical relations on the sets of actor and asset entities, $oe$ is an operation entity, and $taskConditions$ is an expression defining conditions for the profile to be executed. Task conditions describe real-time constraints external to the workflow specification (e.g., contextual factors), or spanning beyond task boundaries, that cannot be expressed on the basis of referenced entities' attributes alone.

Execution profiles are modelled through individuals of the `ExecutionProfiles` WMO class, appropriately linked to `EnhancedEntities` (sub-classes) instances, or, in the case of actors and assets, to logical structures thereof (Figure 7). Task conditions are indicated through the `hasCondition` property, pointing at an `Expressions` or a `LogicalRelations` instance, while the `hasExecutionProfile` property associates a `TaskNodes` instance with its `ExecutionProfiles`. Apparently, execution profiles provide a powerful mechanism for incorporating security policies in workflow models. Profiles reflect authorisation statements, describing conditional variants of an operation's execution, with actor(s), asset(s) and real-time parameters being interdependent. The important role of *entities* should not be neglected here, since they foster

attribute-based constraints on actors, assets and the operation, while workflow variables and purpose consideration provide for Separation of Duty and Binding of Duty (SoD/BoD) [2] enforcement and compliance with the *purpose specification and binding* principle.

Finally, a task is additionally characterised by a *synchronisation behaviour* [3]. In this direction, two synchronisation parameters are defined and implemented by the `isControlSync` and `isDataSync` properties, indicating if a task being the *join* point of multiple incoming control or/and data flows synchronises these flows. Data synchronisation, in particular, comprises a useful mechanism in controlling data linkability; its value determines whether the task processes separately each distinct input and directly passes the result to following tasks ("False"), or waits to receive input from all incoming edges before proceeding with the processing ("True").

Figure 10 shows the actor and asset entities related to the task *DisplayPrescriptionsList,* ontologically defined as `ActorE#3`, `ActorE#4`, `ActorE#5`, `ActorE#6`, `AssetE#3` and `AssetE#4`. The latter two refer to the same data type, therefore in both cases the asset is a patient prescription (`ePrescription`). However, the conditions described by `Expr#3` and `Expr#4` differentiate the prescriptions that can be accessed by the doctor, depending on whether consent has been provided by the subject or not. Specifically, unless the patient has explicitly consented to access of all their medical history, only the prescriptions issued by the currently treating doctor may be shown to the latter, according to `Expr#5`. Indeed, the latter, making use of suitable `Variables` and `VariableBasedAttributes` specifies that the data type `PrescribingDoctor`, defined as an `ePresecription` field through the corresponding `isPartOf` IMO relation, has to be the same with the ID field of the actor of the task in question.

Further, in Figure 11 below, the `OperationEntities` instance `OperE#15` related to a hypothetical anonymisation task indicates, beyond the corresponding IMO operation type, also the anonymisation method to be used, as the value of the parameter `Method` characterising the particular operation (expression `Expr#15`); method *RandomChars* is here assumed to be a registered system function replacing data with randomly generated characters. Moreover, it is specified that the anonymisation will be performed on two specific fields of an `ePrescription`, namely on the `Diagnosis` and `DateOfBirth` assuming that disclosure of these two fields is prohibited, through `AssetE#15` and `AssetE#16`, respectively.

As regards flow of control and data, this is represented through directed *edges* of the corresponding type and, ontologically, as instances of the `ControlEdges` and `DataEdges` classes. Data and control edges share the same characteristics: each connects two tasks and denotes the flow direction, the information exchanged (`carriesInfoEntity`), the underlying conditions (`hasCondition`), under which the transition takes place, and other flow properties (`hasFlowConstraint`); the distinction between them stems from the semantics of the connected operations regarding the manner they receive and consume information. Flow conditions and constraints are both defined as expressions. The former are analogous to task conditions and should hold in order for the transition among two tasks implied by the edge to be performed, supporting, when needed, conditional branching of control or data flow. The latter are conceptually close to the "implementation" attribute of the Send and Receive types of Tasks defined in BPMN [4]; they do not describe elements of the workflow model, but rather low-level properties describing their interaction.

Figure 12 shows the ontological representation of the data edge reflecting the transition from task *DisplayPrescriptionsList* to *EditPrescription*; the information entity constraint `Expr#30` implies that this transition takes place only if the particular prescription has not yet been executed.

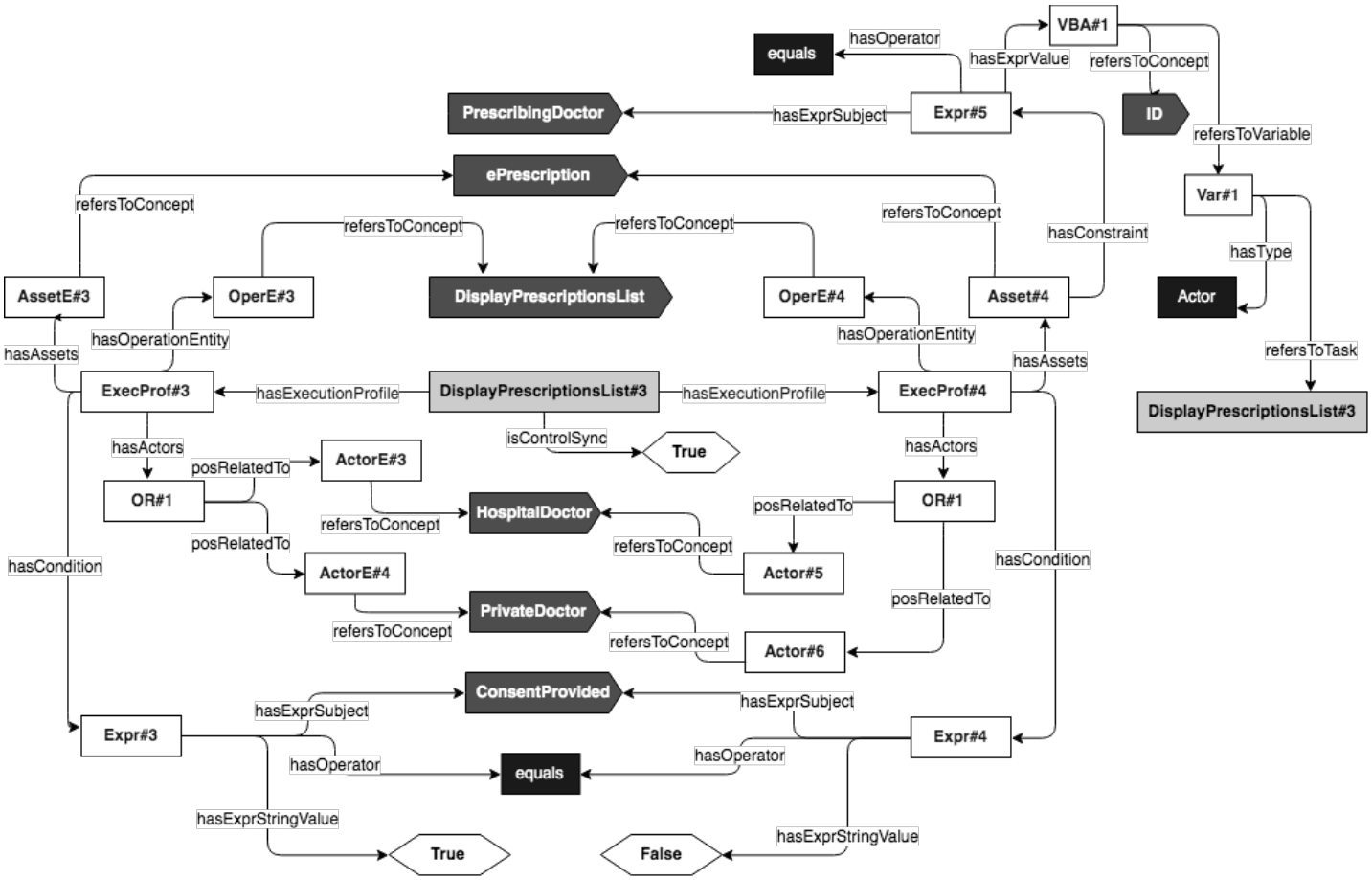


**Figure 10: Ontological representation of the task *DisplayPrescriptionsList***

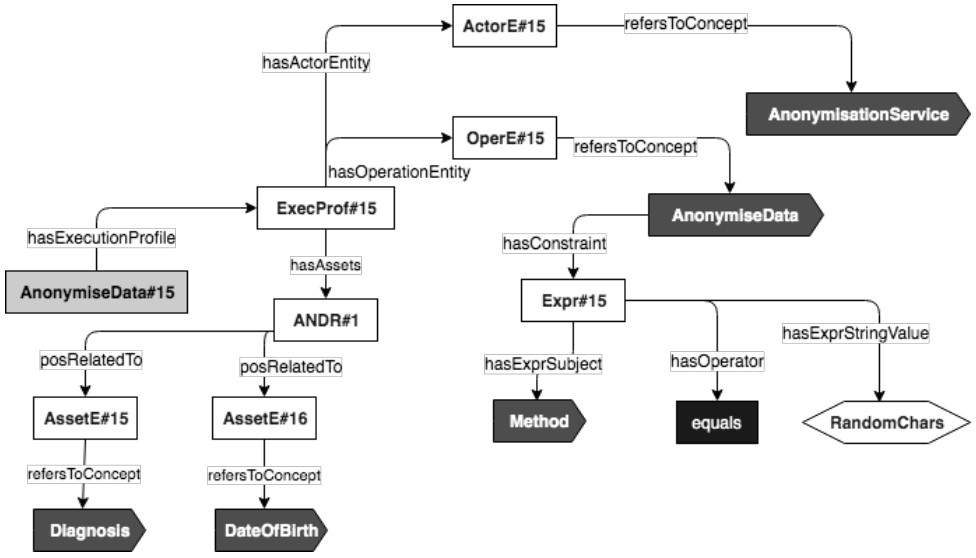


**Figure 11: Ontological representation of an anonymisation task**

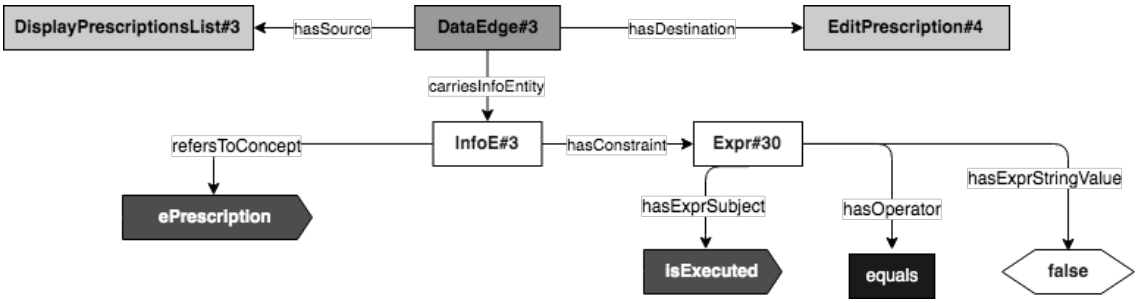


**Figure 12: The edge connecting tasks *DisplayPrescriptionsList* and *EditPrescription***

## 2.3 Process reengineering

### 2.3.1 Compliance directives

The verification of a workflow with respect to privacy compliance is performed based on its ontological representation (cf. Section 2.2) and on a set of *Compliance Directives*, that indicate the terms under which the workflow in question is acceptable. Directives are generated through reasoning over the Policy Model; its main elements are access control *rules*, used for defining *permissions*, *prohibitions* and *obligations* over *actions*, i.e., structures that, similar to tasks, indicate an operation performed by an actor on an asset (for details, please refer to deliverable D3.2 “Initial specification and prototyping of the policy framework”).

Directives creation takes place on the basis of the pairs of all interacting tasks, along with their corresponding interactions (i.e., connecting edges) themselves, what is being referred to as the *Bilateral Associations* (BA) of the workflow. The reason for choosing this pair-wise fragmentation for the initial processing at the level of Directives generation, is that a BA essentially constitutes the elementary unit of flow. Thus, the instructions received are richer in semantics, since it is not only the tasks that matter, but also their interrelations. The main types of Directives considered are:

- *Bilateral Validity Directive (BVD)*: A directive of this type refers to one BA, indicating, for a given purpose and initiator among the specified ones, one valid actor--operation--asset combination for each task and a valid specification of the relationship connecting the two tasks; the latter may refer to the edge as has been defined by the designer, a different edge specification, or even one or more tasks that must be inserted in between the two tasks, so that the control or data flow between them is consistent. All other types of directives presented below refer, in most cases, to a specific BVD, reflecting the fact that requirements and prohibitions may depend on the existence and the different valid specifications of the tasks originally appearing in the workflow.
- *Input Requirement Directive (IRD)*: A task, though being “in principle” accepted, needs to receive some additional input, not included in the BA under consideration. The directive specifies the required input, (optionally) its source, and the task within a valid BA that needs to receive it.
- *Output Requirement Directive (ORD)*: A task specified within a valid BA must provide (some of) its output to a certain task (or structure thereof). An ORD defines the task in a valid BA that must communicate the data, the data themselves and the task structure that must receive them.
- *Task Presence Directive (TPD)*: A task structure must execute, complementing reference BA tasks. If applicable, a TPD also indicates the relative position or data association with respect to the BA task the required one(s) must be found in; for example, a task may require that another has preceded at some point in the workflow.

- *Task Forbiddance Directive (TFD)*: A task must not be executed in the context of a workflow, either at any point or within certain parts of the flow. Each of these directives refers to a task defined by a BVD, specifying the task structure with which the task under consideration is not allowed to coexist, along with their relative position, if applicable.
- *Flow Forbiddance Directive (FFD):* A task is not allowed to have read access to two or more types of information during a single execution instance. Given a valid BA, such a directive prescribes a forbidden additional incoming flow, by specifying the data it is not allowed to receive but also, potentially, a particular task that they must not come from.

All types of directives may optionally be associated with a contextual condition under which the indicated specification, requirement or forbiddance must apply. Furthermore, a precondition or a postcondition may be defined, denoting the fact that said directive is enforceable if a task, or structure thereof, precede, respectively follow. Directives may additionally be characterised by a *compliance profile*, providing for the definition of variations and guiding their proper enforcement. For instance, in a hypothetical TPD, the compliance profile may define that the required task must have been executed *immediately before* the reference task, and, in particular, in a blocking sense, meaning that it must have completed before considered BA executes.

### 2.3.2 Verification methodology

On the basis of the above-described directives, the verification of processes takes place in the Planning Environment, through a procedure briefly presented in what follows.

As said, the directives are generated on the basis of Bilateral Associations (BA). In order for the latter to be extracted, first the workflow model is decomposed to *instance subgraphs* (*IS*); these correspond to the different variants the workflow may take, based on the values assigned to all constraints associated with its flow. That is, when the execution of a task implies conditional branching of the consequent flows based on edge constraints, the mutually exclusive constraint spaces of outgoing edges are separately considered, resulting in an execution tree; its leaves represent the space of instance subgraphs. The concept of instance subgraphs has been often used in workflow science, since it makes easier to handle by breaking up the workflow into manageable components (e.g., [5]).

Based on *IS*, the Bilateral Associations (*BA*) are created, and thereupon verified using the Policy Model, resulting in the set *D* of Directives. The cartesian combination of purposes and initiators (*PIP*) is then reduced to those pairs appearing, according to *D*, to be valid (VPIP), whereas *D* drives the verification of each $is \in IS$.

The first step in the verification of an instance subgraph $is$ concerns the extraction of the different *cases* ($C_{is}$), derived from the Bilateral Validity Directives (BVD) associated with $is$. Each case $c$ reflects an execution variant of $is$, where each task can be executed in a unique manner and all edges between tasks are the ones prescribed by the corresponding BVDs. To make this more clear, for every BA $<t_i, e_k, t_{i+1}>$, each derived BVD comprises a structure $<t_i^*, e_k^*, t_{i+1}^*>$ where $t_i^*$ and $t_{i+1}^*$ incorporate exactly one execution profile each, and $e_k^*$ is the edge appropriately adapted. It is important to stress that $e_k^*$ may include additional tasks mediating $t_i^*$ and $t_{i+1}^*$; this is often the case, e.g., with tasks performing data anonymisation or encryption. Eventually, each case $c$ is a projection of $is$, according to a valid combination of $<t_i^*, e_k^*, t_{i+1}^*>$ structures, derived from the BVDs.

The generation of cases $C_{is}$ is followed by their verification and appropriate transformation, considering also the rest of Directives. In this context, the behavioural *norm* of each task *t* in a case *c* is extracted by the directives

$D_c$ pertaining to the case. Essentially, norms comprise groups of compliance patterns that span across all Directives types and can be verified together for $t$.

*Forbiddance Norms* (*FN*) reflect requirements implied by TFD and FFD. Provisions described by *Direct Norms* (*DN*) concern tasks that should be present in the workflow directly connected with $t$ via an edge, either incoming or outgoing. On the other hand, *Indirect Pre-* (*IPrN*) and *Indirect Post- Norms* (*IPoN*) indicate tasks that should precede, respectively follow, the execution of $t$, with relative position other than direct connection, whereas an *Existence Norm* (ExN) implies the need for a task to exist in the workflow at any position. *State Norms* (*StN*), derived from BVD, reflect requirements related to data state. Finally, norms can be *conditional* or *definite*, depending on whether the corresponding Directives are associated with pre- and/or post-conditions, or not.

All tasks comprising the case are verified against the associated norms. Therefore, tasks are topologically sorted [6], providing for both forward and backward traversal, and the application of the norms for the progressive transformation of the case $c$ takes place, resulting to its verified version $vc$ (or to failure). The procedure begins and finishes with the application of forbiddance provisions; the reason is that, on the one hand, the case may be rejected at the very beginning due to some conflict implied by *FN*, while, on the other hand, checking against forbiddances is deemed necessary following any transformations that may have happened due to the application of the other types of norms.

The latter takes place in three phases; first, the definite provisions are applied, followed by the conditional ones. In each phase, direct norms precede indirect pre- and post- norms; the reason why indirect norms are not applied together, as is the case with direct, is that post- norms require traversing the tasks of the case in a backward manner. Third, norms related with data state are applied, in order to perform the corresponding verification and transformation after all other norms have been applied and, consequently, all task additions and flow modifications they imply have already been enforced.

After this loop has been executed over all cases $C_{is}$, the cases $VC_{is}$ found to be valid are being merged, providing the verified instance subgraph $vis$; merging concerns the aggregation of the tasks representing the same activity in the different cases, and the unification of the corresponding edges. Similarly, when verification of all instance subgraphs is complete, the verified ones ($VIS$) are merged providing the final verified workflow model $WM_V$. In other words, similarly to the decomposition of the initial workflow model to instance subgraphs and cases, the final $WM_V$ is assembled from its elementary parts, i.e., its cases and verified instance subgraphs, into a unified specification. Intuitively, in order for the workflow verification to be successful, there should be at least one verified case $vc$ resulting from the procedure.

The basic scheme summarised above has some variants concerning mostly the repetitive execution of certain parts, in order for a case, a subgraph, or the model as a whole, to be verified again; this is in order to capture potential privacy flaws that the modification may have introduced. For instance, two new tasks, introduced during verification, may conflict with each other, which cannot be captured by the initial directives. Hence, repetition of some procedures is necessary, until the workflow “converges” to a definitive structure.

### 2.3.3 Process reengineering by example

With reference to Figure 4 and following the above-presented verification procedure on the basis of the corresponding directives, the workflow specified by the designer is transformed by the Planning Environment into its GDPR-compliant equivalent (Figure 13), with the following modifications[3]:

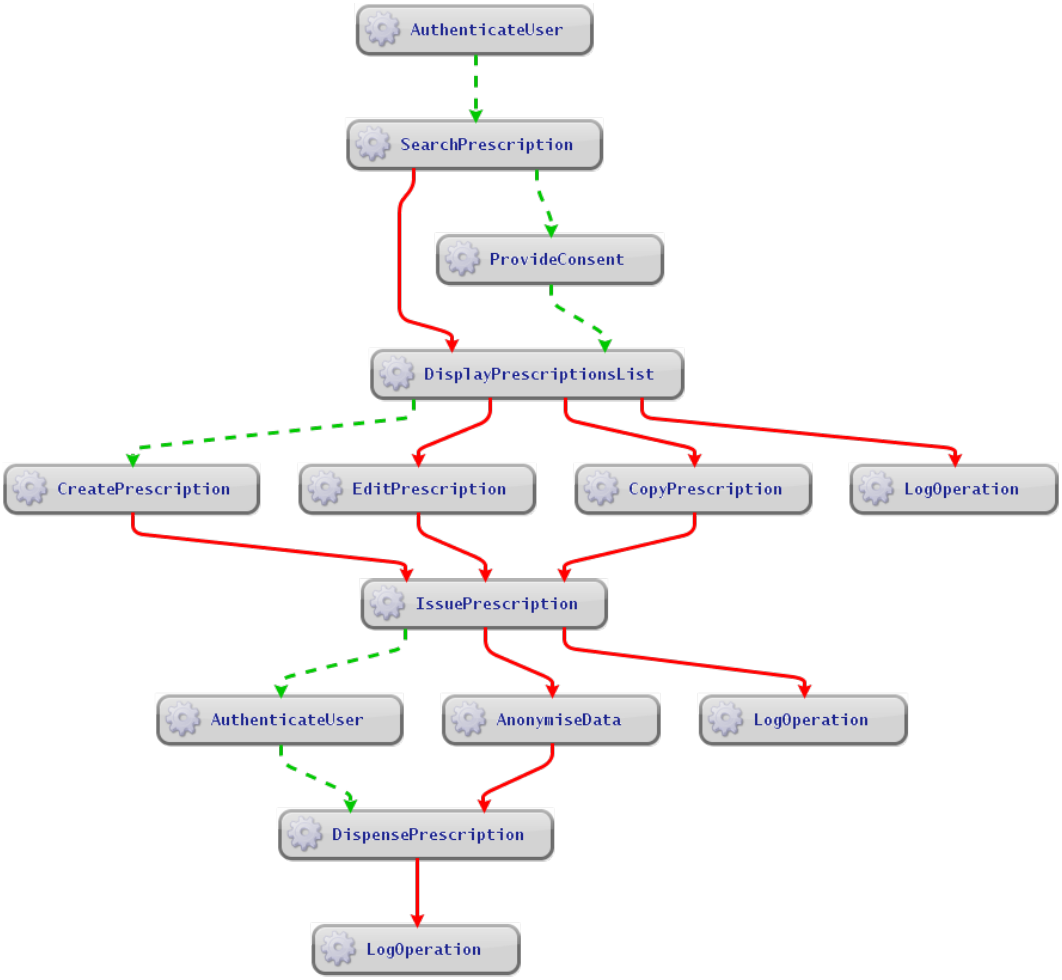


**Figure 13: Final verified workflow model**

- The purpose `MedicalDiagnosis` is removed from the set of purposes that the particular workflow is permitted to serve. This means that according to the organisational policies, the tasks and information exchanges contained therein are not compatible with such purpose, hence the workflow will not be executed in case the initiator declares this as intended purpose. The only allowed purpose remaining is `PreventiveMedicine`.
- The task `DisplayPrescriptionsList` is linked to two execution profiles, denoting the fact that individuals holding either role `HospitalDoctor` or `PrivateDoctor` are authorised to perform it on all prescriptions without discrimination only when consent has been provided; otherwise the doctor is allowed to access only those prescribed by themselves. This is due to the four BVDs derived for the BAs involving the task `DisplayPrescriptionsList`, that provide two different valid

[3] Note that in the particular Planning Environment GUI screenshot the specifics of tasks and edges are not depicted for clarity reasons.

specifications for it, for the purpose of `PreventiveMedicine` and each of the initiators, leading to the formation of two different cases for each of the resulting instance subgraphs (cf. Figure 10).

- The task `AnonymiseData` is inserted, as prescribed by the BVD connecting the tasks `IssuePrescription` and `DispensePrescription`, for preventing read access to selected fields of prescriptions processed by the pharmacist (cf. Figure 11). Notably, the planning system in this case automatically complements the description of the information entity reaching `DispensePrescription` with state information in order to keep track of how the ePrescription is affected with respect to anonymisation-related processing. This is defined for the corresponding data edge as shown in Figure 14.

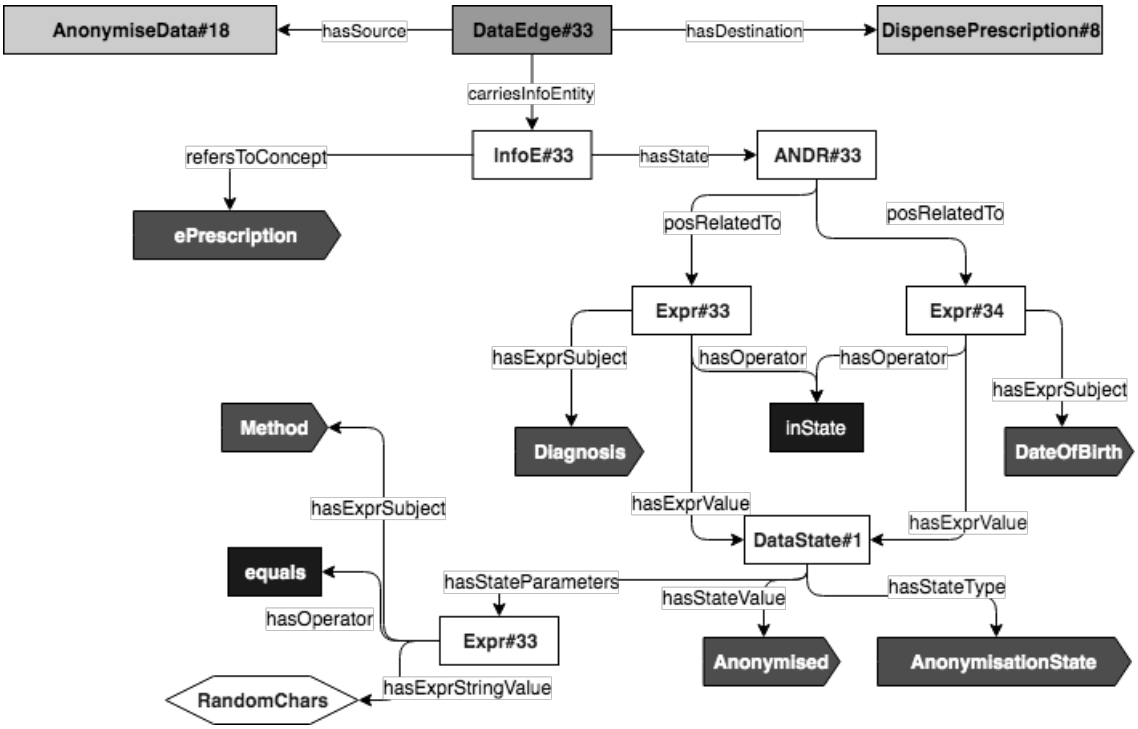


**Figure 14: The information entity along the path *AnonymiseData →DispensePrescription***

- Three instances of the task `LogOperation` are inserted for accountability purposes right after tasks `DisplayPrescriptionsList`, `IssuePrescription` and `DispensePrescription`, defining the latter in each case as their assets; this is required by ORDs linked to all BVDs involving the three tasks and is enforced during Direct Norms application.
- The task `AuthenticateUser` is inserted before prescription dispensation, in order to ensure that also the system has indeed authorised the pharmacist to perform the declared task.
- The edge leading to task `EditPrescription` is complemented with the constraint that the prescription has not been executed (cf. Figure 12).
- The task `CopyPrescription` is complemented with the constraint that the specialty of the doctor initiating the process and executing the task is the same with the specialty of the prescribing doctor.
- The task `ProvideConsent` is inserted, to ensure that the corresponding patients (data subjects) have been asked to give their consent for processing their medical records. This is the result of the enforcement of an IRD, again linked to all BVDs, during Direct Norms application for the task `DisplayPrescriptionsList`. The compliance profile of the particular directive specifically has

the value "path-binding", indicating that the required task must receive its input (e.g., the ID of the patient the prescriptions concern) from a task along the data path providing analogous information to `DisplayPrescriptionsList`, i.e., `SearchPrescription`. Further, the task `DisplayPrescriptionsList` is annotated with `isControlSync` to be "True", so that it cannot execute without the reception of consent (cf. Figure 10). Finally, the prescribed specification for `ProvideConsent` requires that its actor must be the owner of the prescription being processed by the task `DisplayPrescriptionsList`, i.e., it has to be the patient affected by the prescription that provides their consent, implying a form of BoD. Moreover, the consent must explicitly concern the purpose for which the entire process is executed, something that can be defined through the parameters of the corresponding operation. The ontological representation of the task `ProvideConsent` is presented in Figure 15, showcasing, among others, the use of workflow variables.

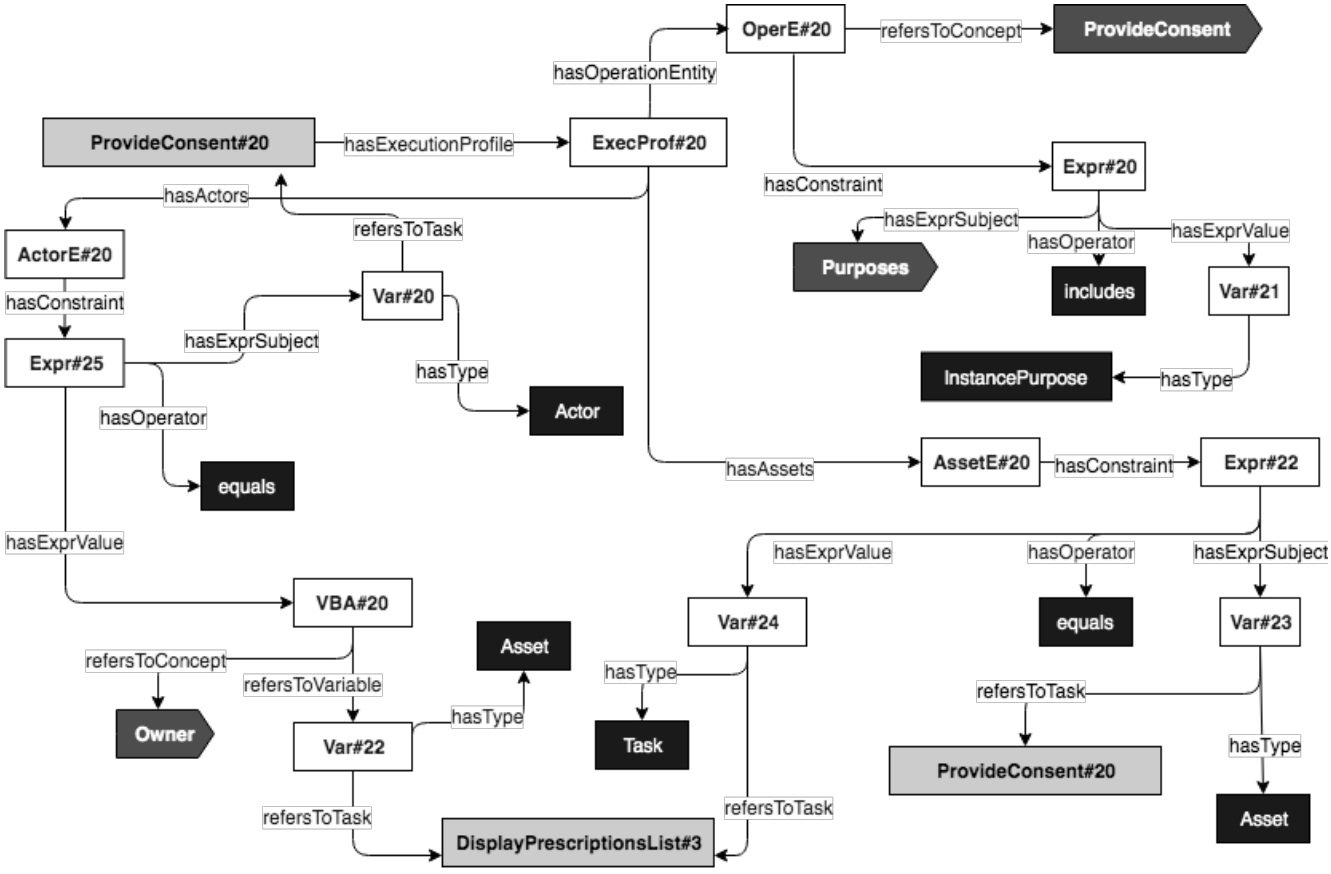


**Figure 15: The task *ProvideConsent***

## 2.4 Planning environment

This Section outlines the Planning environment, which is the software system providing all the necessary functionality for both process modelling and process re-engineering, as have been documented, respectively, in Sections 2.2 and 2.3. Specifically, Section 2.4.1 describes the overall software architecture, whereas Section 2.4.2 provides insights to the process modeller, i.e., the user interface of the planning environment.

### 2.4.1 Software architecture

In Figure 16, the logical architecture of the Planning environment is sketched. As illustrated, it internally consists of the following modules:

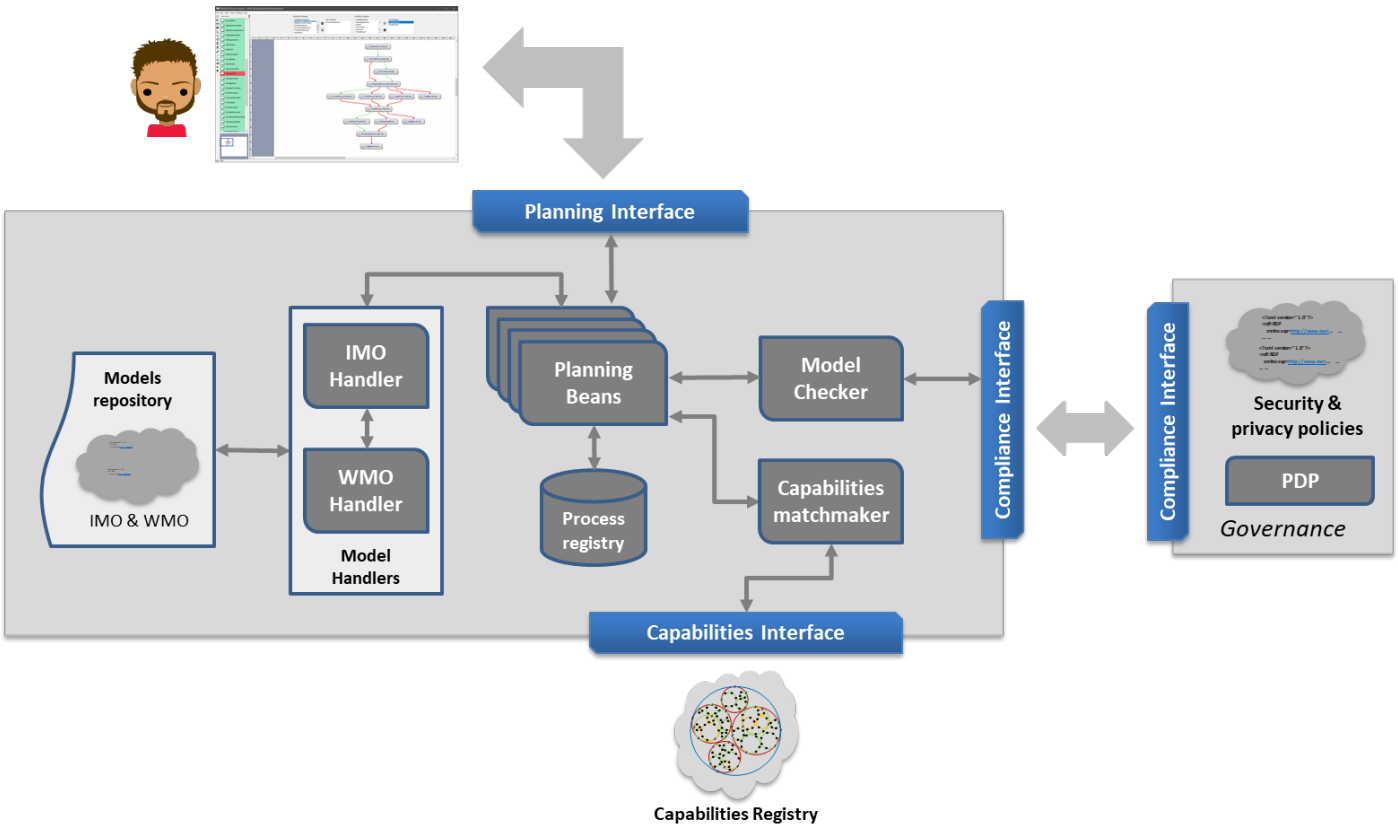


Figure 16: Planning environment logical architecture

**Planning Beans**. Planning constitutes an essential part of a process management system; the Planning Beans, constitute stateful entities managing the transactions between the Planning environment and its client side, such as the user interface described in Section 2.4.2, and maintaining their state, during the process design phase. In essence, each Planning Bean is assigned with the management of the specification, verification and transformation of one process; that is, it is in charge of all the stages in a process lifecycle until it becomes GDPR-compliant.

**Model Checker**. It implements the core planning intelligence, being the entity that is invoked by the Planning Bean managing some process, in order for the latter to be verified and appropriately re-engineered for becoming compliant. To this end, it performs the procedure described in Section 2.3.2, guided by the BPR4GDPR governance module by means of Compliance Directives that are retrieved through the Compliance interface (see below).

**Models repository**. This module provides hosting to the two fundamental ontological models used in the context of process modelling and re-engineering, i.e., the Information Model Ontology and the Workflow Model Ontology. To this end, it constitutes an RDF Triplestore, undertaking all functions regarding the management of ontologies.

**Model Handlers**. This module encapsulates the ontologies, providing a convenient API that makes the underlying models— stored in the Models repository, described above— accessible by the other components, particularly the Planning Beans and, through the latter, the Planning Interface (see below) and client-side components. In this context, it offers all functions necessary for retrieving information from the models, as well as for process modelling implementation leveraging semantic ontologies as described in Section 2.2.

**Capabilities matchmaker**. It identifies and verifies, through interaction with the Capabilities Registry, the capabilities that are made available by the BPR4GDPR run-time environment. These capabilities are leveraged by the Model Checker for identifying the functions offered by the underlying infrastructure, in order to accordingly adapt the process models. For instance, the introduction of an anonymisation function in a process model depends on whether this function is indeed offered by a tool or not.

**Process registry**. It keeps track of the registered processes, along with their metadata (process ownership, history, etc.), providing a repository thereof.

In order to interact with the other BPR4GDPR components, the Planning environment provides three interfaces, devised, respectively, for creating and managing process models, requesting compliance-related information from the BPR4GDPR governance module, and querying about the capabilities of run-time components.

**Planning interface**. It constitutes the central API for interacting with the Planning environment. Through this interface, the appropriate functions are provided for the creation and management of process models, including their re-engineering towards becoming compliant.

**Compliance interface**. This interface is devised to interact with the BPR4GDPR governance module (cf. Deliverable D3.2 “Initial specification and prototyping of the policy framework”), in order to be provided with the Compliance Directives (cf. Section 2.3.1). To this end, the governance module implements a counterpart interface.

**Capabilities interface**. This interface is used for querying the Capabilities Registry[4] for the set of available capabilities provided by the various tools integrated into the BPR4GDPR run-time environment.

### 2.4.2 Process modeller

For the administration of the planning procedure, a user-friendly software application has been developed, devised for the design of processes and automation of processes re-engineering. This tool, the *modeller* (Figure 17), provides all necessary functionality for the specification of tasks and flows, including actors, operations, assets, flow of information and all other aspects of a process, as well as the corresponding ontology, as described in Section 2.2. The use of the modeller hides the technical details of the process model, requiring no particular technical expertise by its users; it translates the input provided through the graphical interface to machine code.

[4] The Capabilities Registry is a component provided in the context of BPR4GDPR WP5 “Compliance Toolkit”; it will be documented in the Deliverable D5.1 “Initial specification and prototyping of the compliance toolkit”.

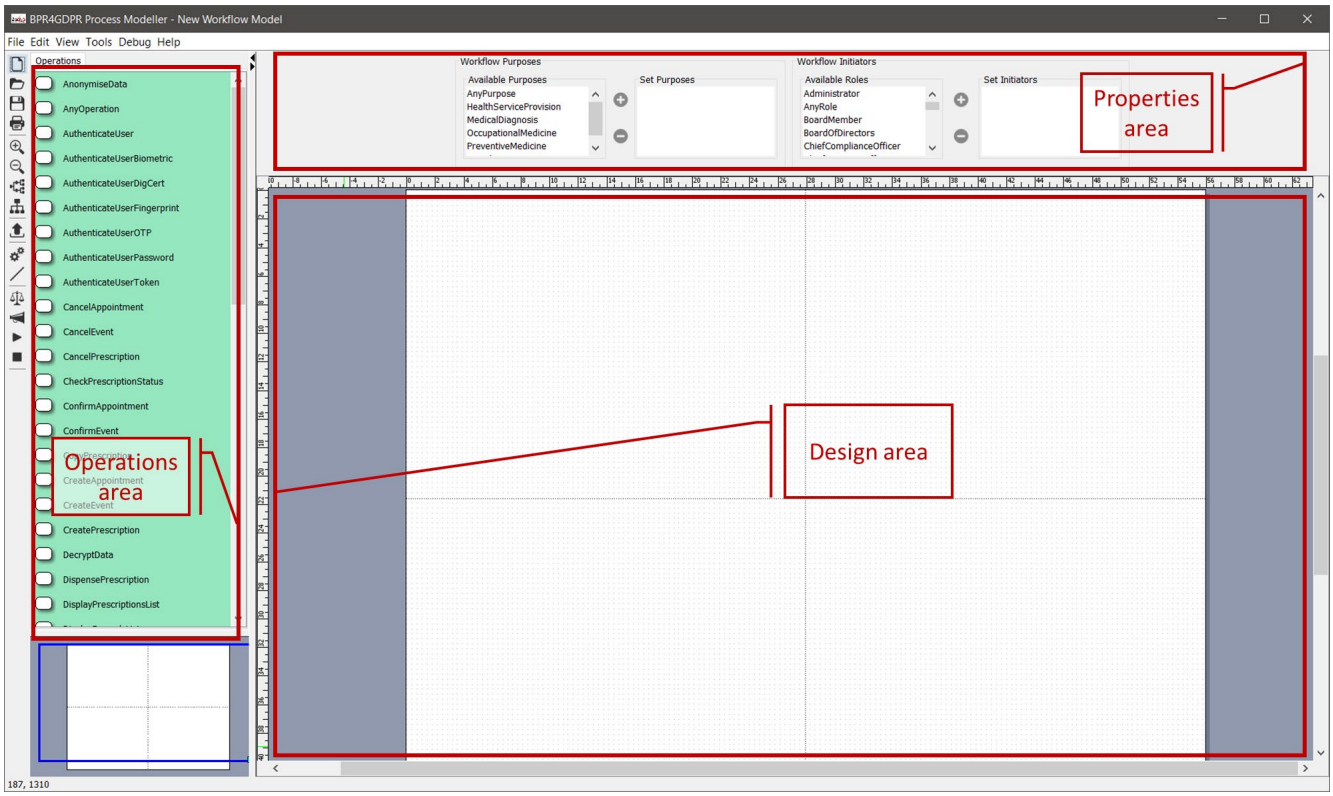


**Figure 17: Process modeller**

As illustrated in Figure 17, there are three main areas the user sees upon logging in, specifically the *Design*, *Operations* and *Properties* areas. The Design area allows graphically modelling a process, notably its tasks and flows, along with all related aspects (actors, assets, etc.). The Operations area provides the list of available operations which are retrieved from the corresponding `Operations` class of the Compliance Ontology, sorted alphabetically for facilitating navigation; in order for the user to create a new task in the process model, all that needs to be done is to drag and drop the operation to the Design area. The Properties area is devised for the definition of a process model's properties, including its association with sought purposes and the identification of the roles that may serve as the initiators of the process. The user interface is complemented by some control functions, particularly the buttons located at the left side of the Operations area, as well as a *map* for making easier the navigation in the model.

In Figure 18, the user has already defined a process, actually the process of Figure 4, illustrated at the Design area. The figure focuses on the definition of purposes and initiators; as shown in the highlighted area, the user defines these concepts by selecting and de-selecting items from respective lists, leveraging "plus" and "minus" buttons. As regards the population of the lists, their items are retrieved from the `Purposes` and `Roles` classes of the Compliance Ontology. In the example of Figure 18, two purposes have been selected, notably `PreventiveMedicine` and `MedicalDiagnosis`, whereas the selected initiators are `HospitalDoctor` and `PrivateDoctor`.

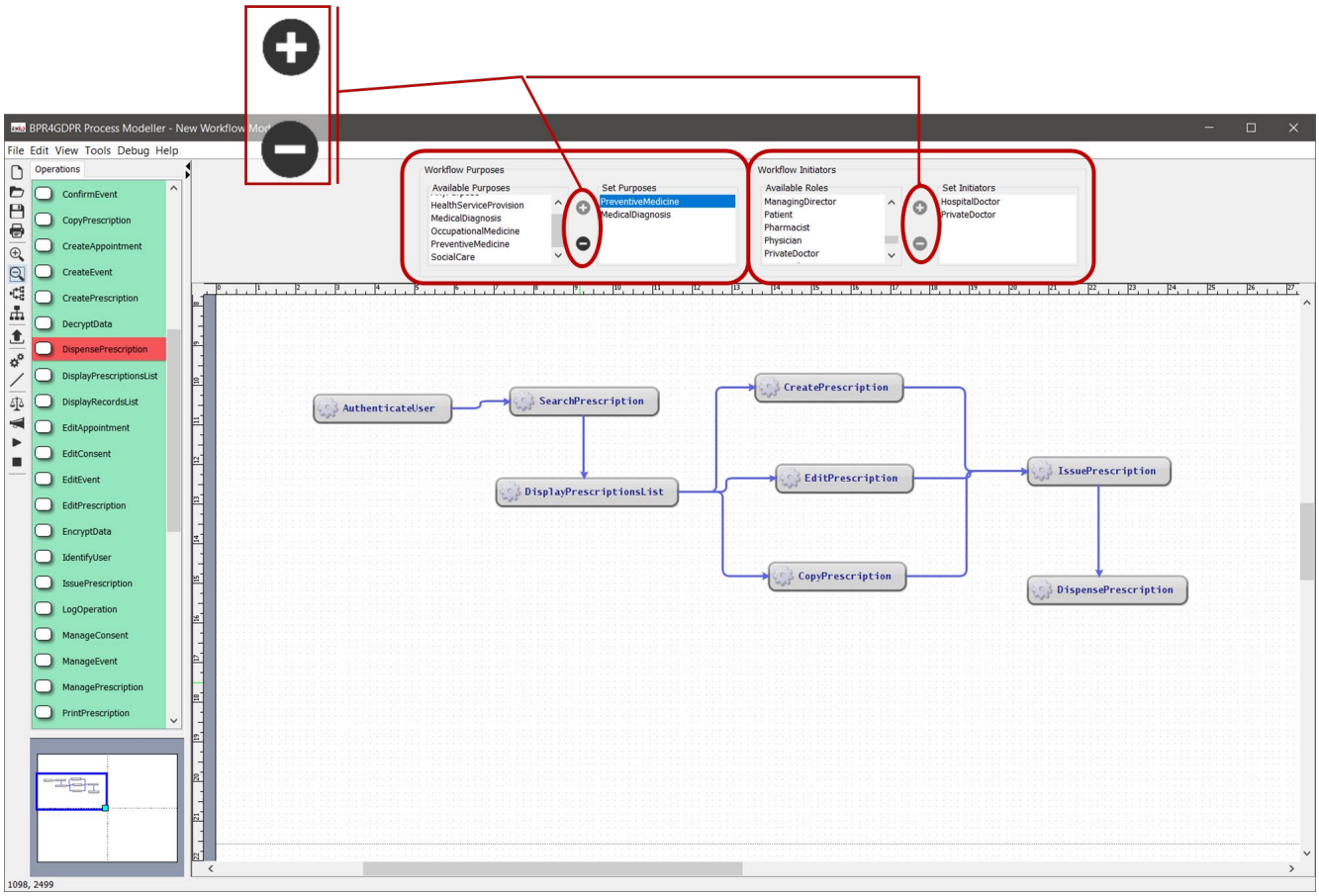


**Figure 18: Definition of purposes and initiators**

As said above, tasks are created by dragging and dropping operations from the list of the left side of the modeller. For connecting tasks, i.e., defining the control and data flows among them, the user simply wires the boxes representing the tasks in the Design area. It should be noted that such edges, upon their creation are neutral as regards their type (i.e., control or data flows). The user has the option to define the flow of type, leveraging a context menu, as shown in Figure 19; in any case though, the flow type will be automatically determined as part of the re-engineering procedure.

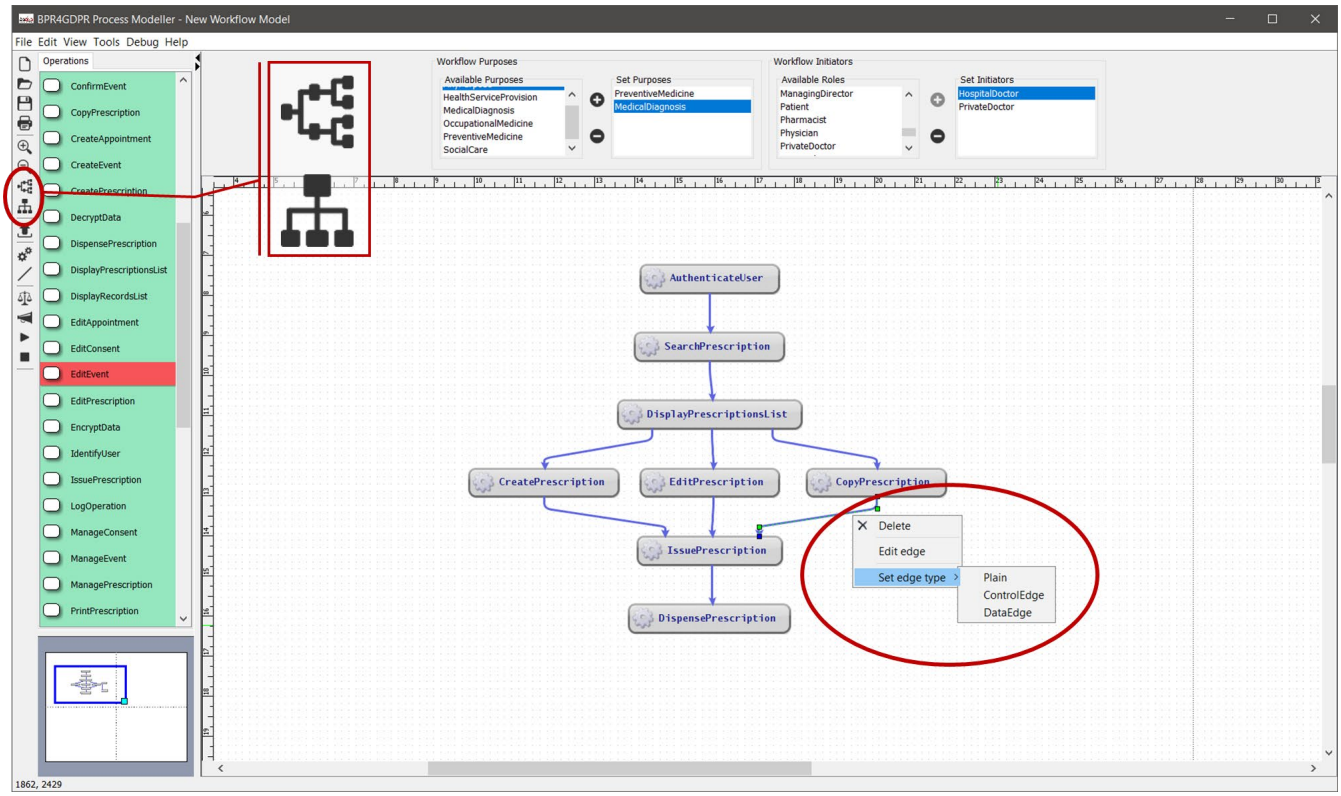


**Figure 19: Flow type context menu and automatic alignment**

Another functionality illustrated in Figure 19 is that of automatic graphical alignment of the process model, in order to become more user-friendly. There are two types of alignment, vertical (already applied in the process model of the figure) and horizontal, applied by means of the corresponding buttons highlighted in Figure 19.

The context menu of an edge is also the means for the specification of what data types the edge shall carry, as highlighted in Figure 20. Upon the selection of “Edit edge”, a pop-up window enables the user to define the types from a list that is retrieved from the `DataTypes` class of the Compliance Ontology, along with any constraints, e.g., for purposes of data filtering. As shown, in Figure 20 the user has selected `ePrescription` as the data type to be handed over from the `DisplayPrescriptionsList` task to `EditPrescription`.

Similarly, a user may leverage the context menu of a task, in order to define essential aspects of the task, notably actors, assets and parameters. This is illustrated in Figure 21, where the user is assumed to have selected `ePrescription` as the intuitive asset of the `EditPrescription` task.

The most important functionality provided through the modeller is process re-engineering, towards compliance. As illustrated in Figure 22, a dedicated button triggers the execution of the underlying procedure (cf. Section 2.3.2) that results to a compliant process. As Figure 22 shows, all modifications described in Section 2.3.3 are visible in the resulting process.

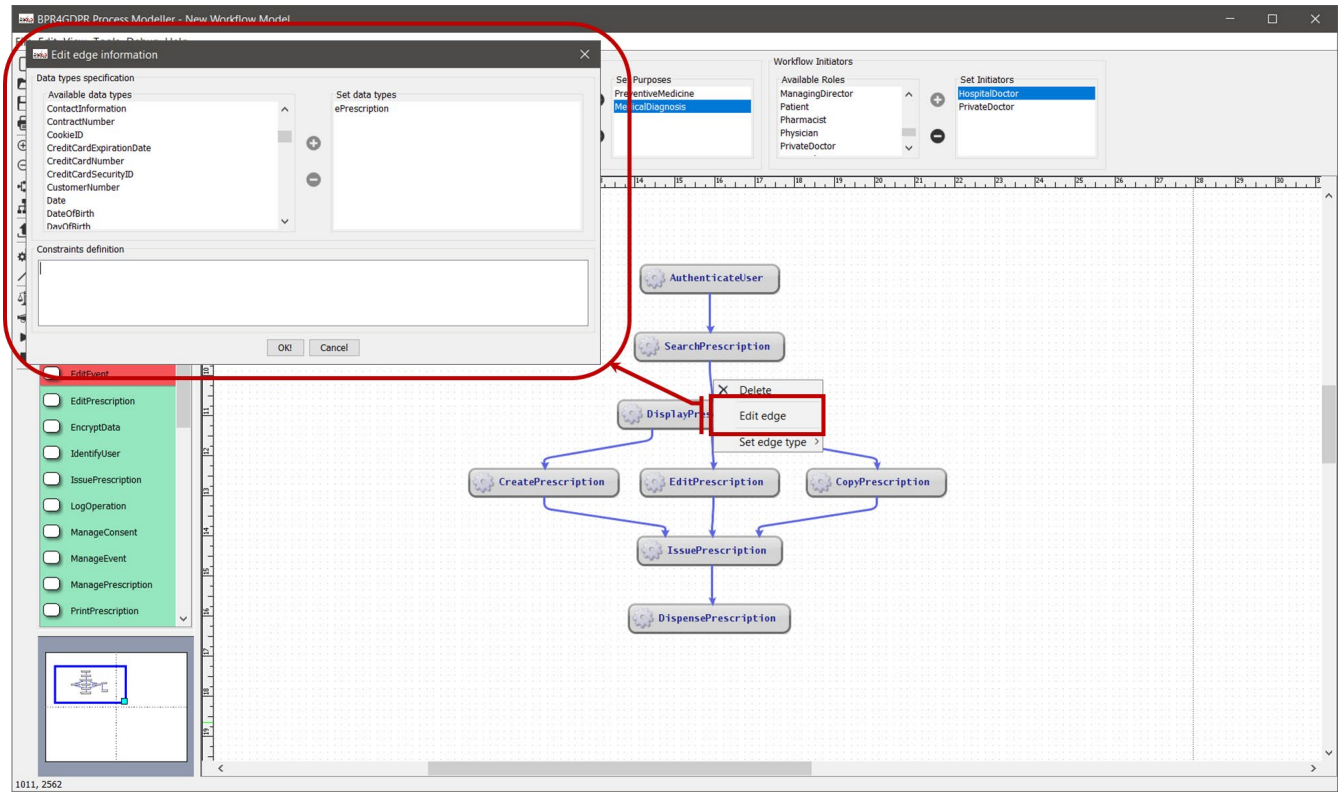


**Figure 20: Definition of edge-related information**

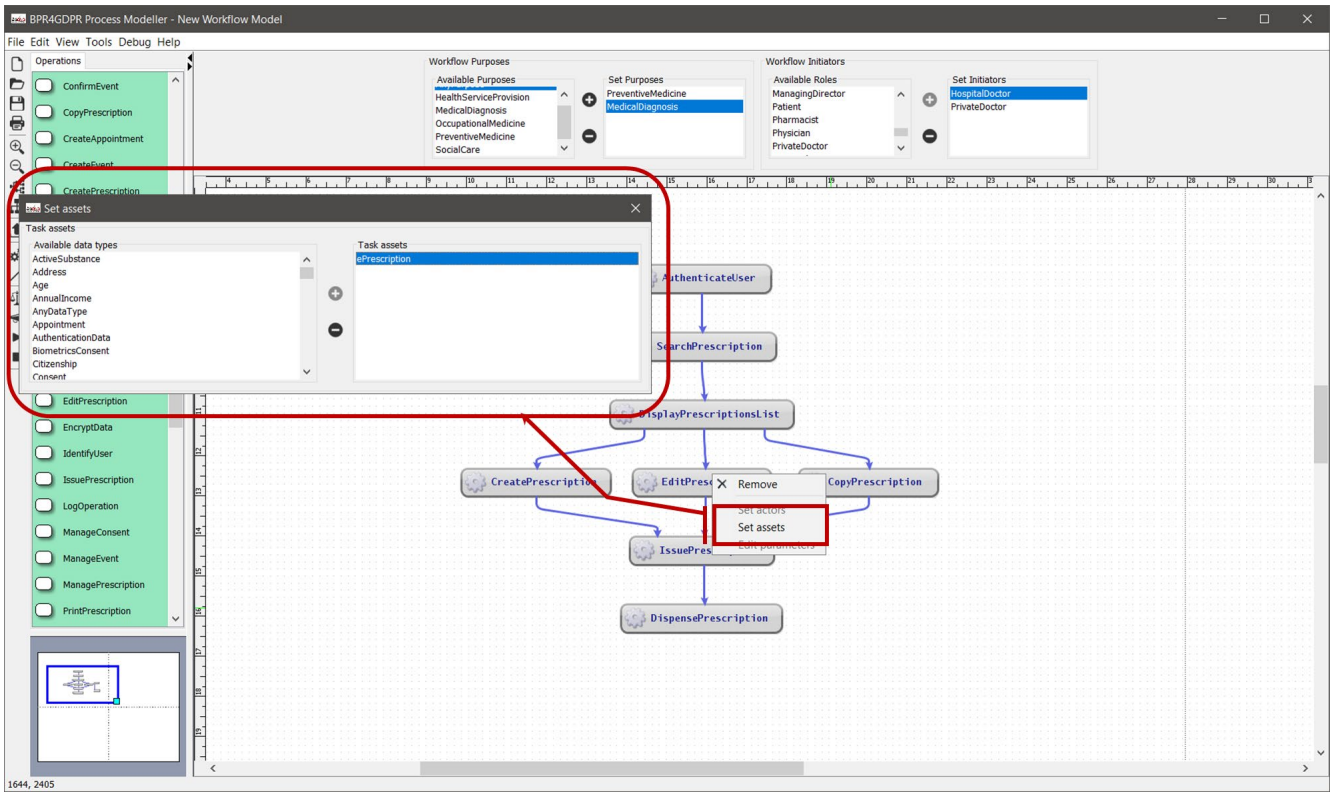


**Figure 21: Task assets' definition**

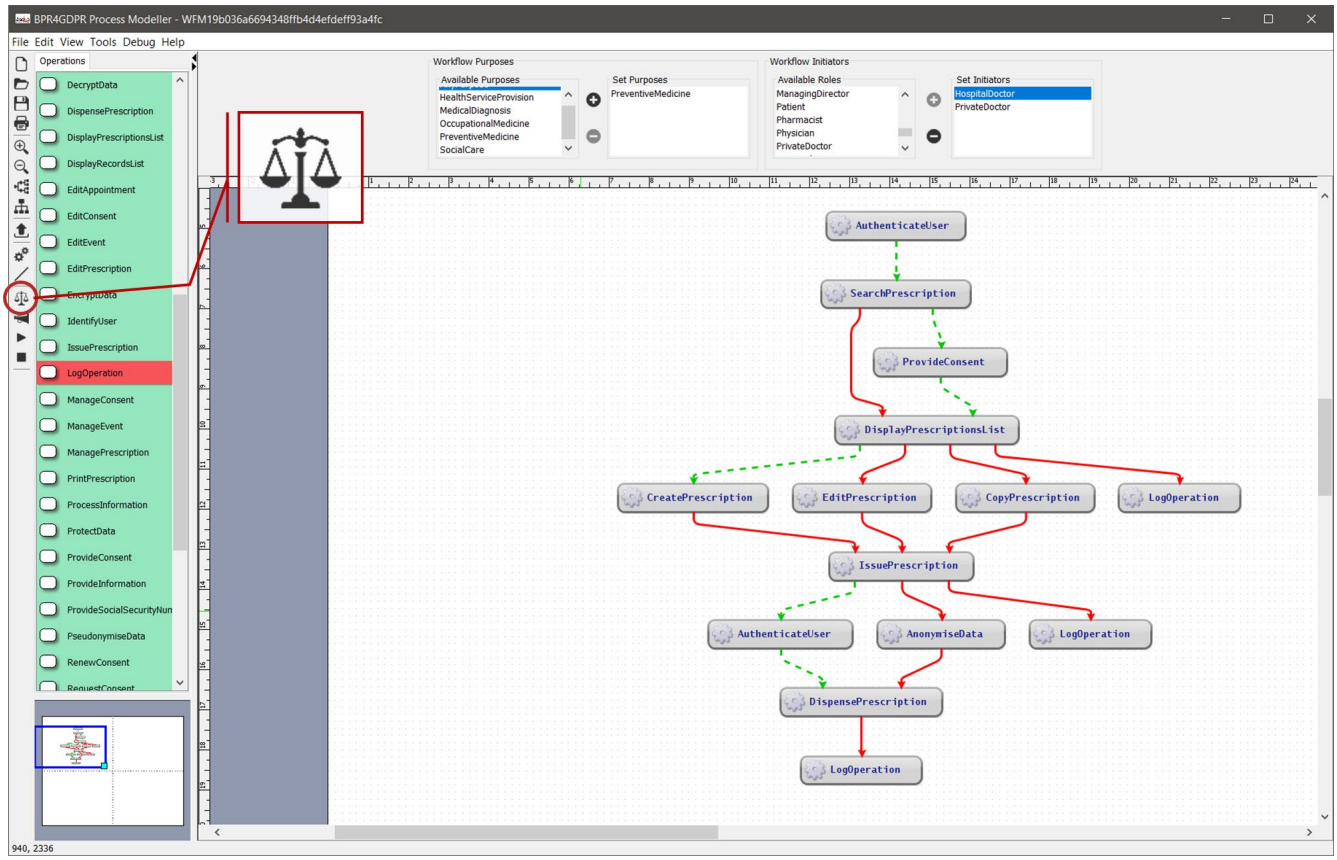


**Figure 22: Process re-engineering**

# 3 Business Process Reengineering Based on Business Process Mining

Business process reengineering (BPR) is an approach for change management in which the related tasks are radically redesigned. The important goal of BPR is to analyze workflows within and between enterprises in order to optimize processes to achieve the goal of improving product/service output, quality, or reducing costs.

Typically, this approach involves analysis of organisational workflows and identifying the necessary changes, finding processes that are inefficient, and figuring out ways to adjust or change them. Generally, a BPR plan lifecycle includes the following stages [7]:

- Identifying the processes for the necessary changes
- Analyzing As-Is process
- Designing To-Be process
- Gap analysis and implementation of re-engineered process
- Monitoring business processes flow and improving them continuously

Process mining techniques provides organizations with leverage in accomplishment of different phases of BPR plan, which is the case with the GDPR compliance of business processes as well.

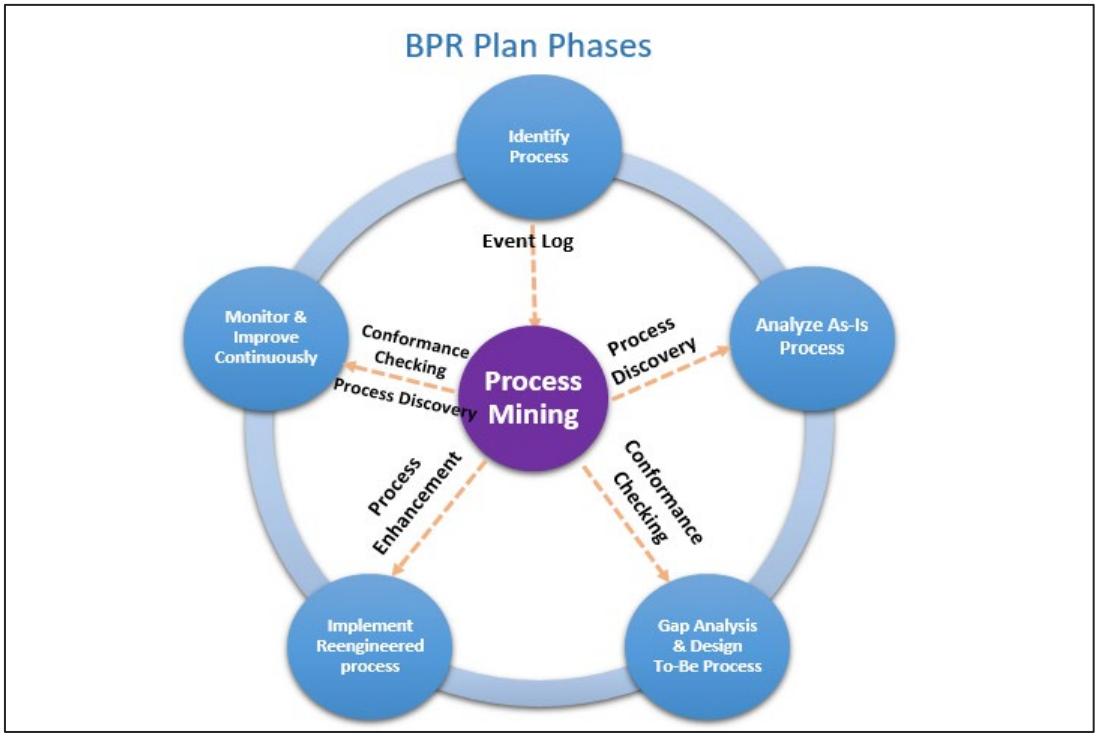


**Figure 23: Business Process Reengineering Based on Business Process Mining**

Usually, process mining is performed to discover, monitor and improve real processes (not assumed or modelled processes) and is primarily based on the knowledge extracted from the event log of information systems. Typically, there is a gap between a modelled process and the trend expected to occur in the process execution and what occurs in reality. The goal of process mining is to discover and decrease problems caused by these deviations. Figure 23 shows how various process mining techniques can be utilized in different phases of a BPR plan. As shown in figure 24, process mining techniques and algorithms are classified as three main categories namely: discovery, conformance checking, and enhancement.

1. Process discovery is the most common process mining technique and can automatically detect and generate the process model based on event logs and reality [8] and extract statistical information about process characteristics. In case of existence, discovered process model provides information about inconsistencies, bottlenecks and loops. The output of this technique can be used for As-Is analysing phase in the BPR Plan. In addition, during the monitoring phase of the BPR plan, process mining mainly takes a very important part. Applying process discovery technique in this stage can show real execution and flow of redesigned processes and facilitate monitoring and evaluating of KPIs.
2. In conformance checking, a predefined/discovered process model is compared with execution traces of the event log. The goal of this technique is to assess and quantify the harmonisation between the process model and the event log [9]. This technique can be applied for ascertaining different aspects of the process, such as compliance with organizational vision, business rules and policies. In the BPR4GDPR project context, conformance checking will be applied to investigate the compliance of the organizational processes to the GDPR and discover deviations, if any. The results help the organization to distinguish the exact points of the process that need improvement to become GDPR-compliant [12]. Furthermore, applying this technique facilitates process monitoring in the stage of monitoring and improvement phase of the BPR Plan.
3. The third type of process mining is enhancement in which current processes can be improved or extended using additional information contained in the event log like resources, case attributes, data attributes [8].

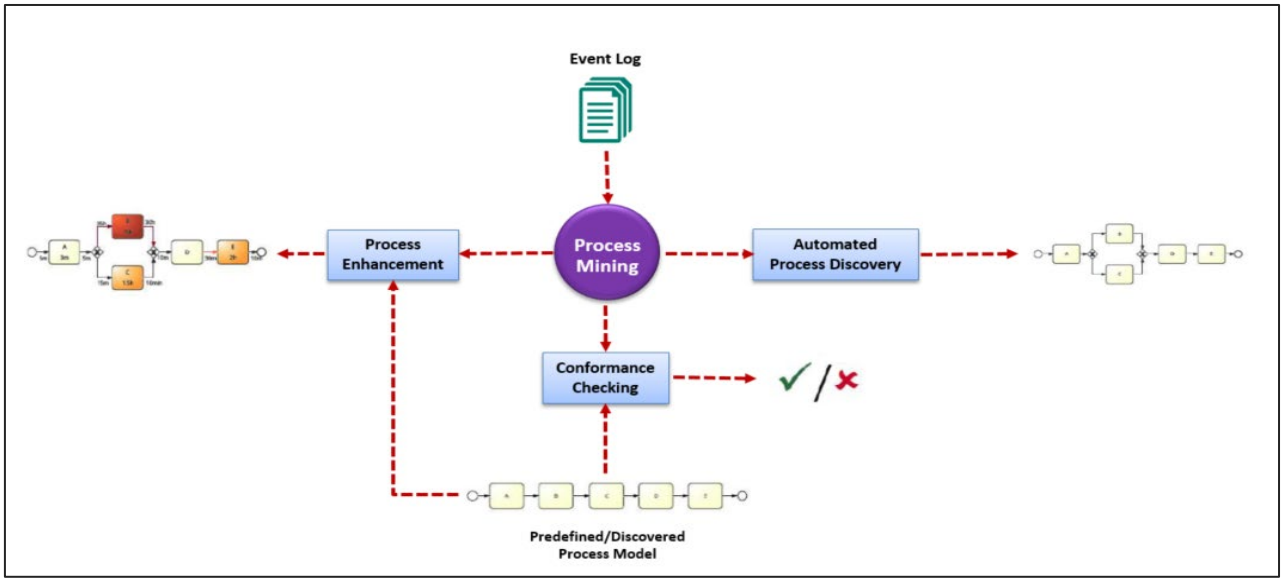


Figure 24: Process Mining Techniques

In summary, process mining can be employed to perform complex tasks of business process reengineering. These techniques not only identify processes and build their formal models but also check compliance of the models with real situation and determine deviations by comparison of event data with process model [10]. It supports decision making and gives recommendations to improve processes. In the next section we introduce the initial specification and prototyping of the process re-engineering framework based on process mining techniques.

# 4 The Initial Prototyping of the Reengineering Framework (TUE)

This section provides overview and specification of the process discovery and mining tool of initial prototype of the process reengineering framework. The prototype tool is demonstrated with the help of a car dealership use case in the light of Right to Erasure provision (usually referred to as Right to be Forgotten or RTBF in short) of the GDPR.

## 4.1 Right to be forgotten (RTBF) use case:

Article 17 of the GDPR titled "Right to erasure ('right to be forgotten')" states that:

> *"The data subject shall have the right to obtain from the controller the erasure of personal data concerning him or her without undue delay and the controller shall have the obligation to erase personal data without undue delay……."*

Right to be Forgotten (hereafter referred to as RTBF) requires data controllers to erase the related personal data of a data subject in case the data subject formally makes such a request, except some circumstances mentioned in the Article 17. In addition, Article 12 of the GDPR requires data controllers to provide information on action taken in response to requests made by data subjects in the context of Articles 15 to 22. In spirit of the Article 12, the data controller therefore should notify/acknowledge the data subject after the erasure request has been acted upon and fulfilled.

The use case being considered for this prototype is a modified version of CAS car dealership use case. The abstract business process model is provided as Figure 25. In the process, the car dealership acquires potential leads data from its business partner(s) and process this data for identifying quality leads. The quality leads are contacted by the car dealership for scheduling a test drive. Leads may either schedule a test drive which may or may not turn into a business deal or they may simply deny scheduling a test drive. In either case the process terminates.

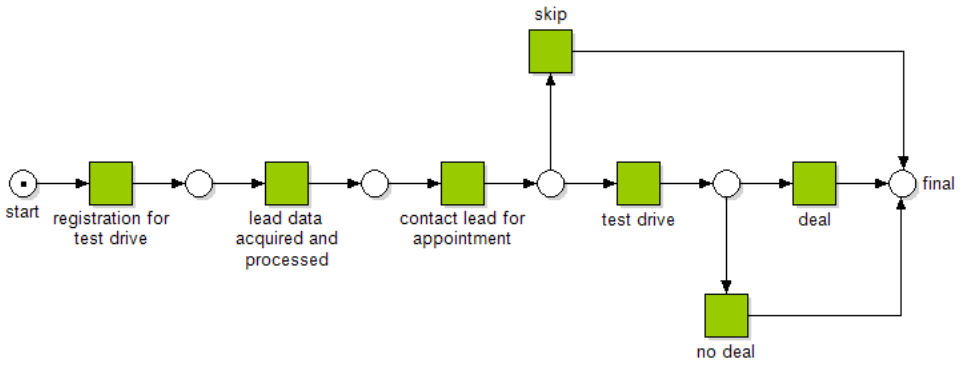


**Figure 25: Car dealership use case**

Leads are entitled to request erasure of personal data. To be GDPR-compliant, car dealership, in majority of cases, must delete the data and acknowledge the lead accordingly. In terms of business processes, once an erasure request has been received it should be immediately followed by a data deletion action and thereafter a notification action shall take place as part of the business process. This behaviour shall not only be reflected in the business process model but also in the process execution. The tool explained in the next subsection will ascertain if the activities have been performed and if also in the right order.

## 4.2 Overview of the process discovery and mining tool of the reengineering framework

Referring to Figure 26, the initial prototype of the envisaged process discovery and mining tool as part of the reengineering framework provides two-fold functionality: process discovery and enactment for realization of the business process in execution, and compliance checking of business processes against RTBF provision of the GDPR. Business process model and event log are the two first class citizens of process mining. For the present functionalities of the prototype only an event log is sufficient. The event log can be extracted out of the information systems in required *xes* format through ETL tools (Extract, Transform, Load) or alternatively as common *csv* format. The *csv* format event log can be converted to required *xes* format by the Convert CSV to SEX plugin available in the prototype. The compliance checking functionality additionally requires a formal specification of the RTBF.

### 4.2.1 Process discovery

Two process discovery plugins are provided as part of the prototype. The only input required is an event log. One plugin provides the straight-forward functionality to discover a process model (as Petri net) out of the observed behaviour of the business process in the form of the event log. Another plugin along with discovery of a process model (as process tree), provides functionality to replay the event log on top of the process model. Replaying event log over process model provides a realization of the execution of the cases in time domain. Deviations in execution of the process from the respective process model are visualised as well. Visualization of the discovered model at different abstraction levels is possible.

### 4.2.2 Compliance checking

Compliance checking plugin, provides functionality to uncover cases deviating with respect to the Right to be Forgotten. Along with event log, the plugin requires a suitable and formal specification of the rule (RTBF in this case). The compliance plugin confronts the event log with the formal specification of the RTBF (a Petri net model in our tool) and discovers the non-compliant cases in the event log. The compliance diagnostics/statistics are visualized in an interactive environment. Along with the compliance/non-compliance statistics, the compliant and non-compliant cases can be filtered in/out. These filtered cases can be further investigated by process experts and stakeholders for investigating reasons causing these non-compliances.

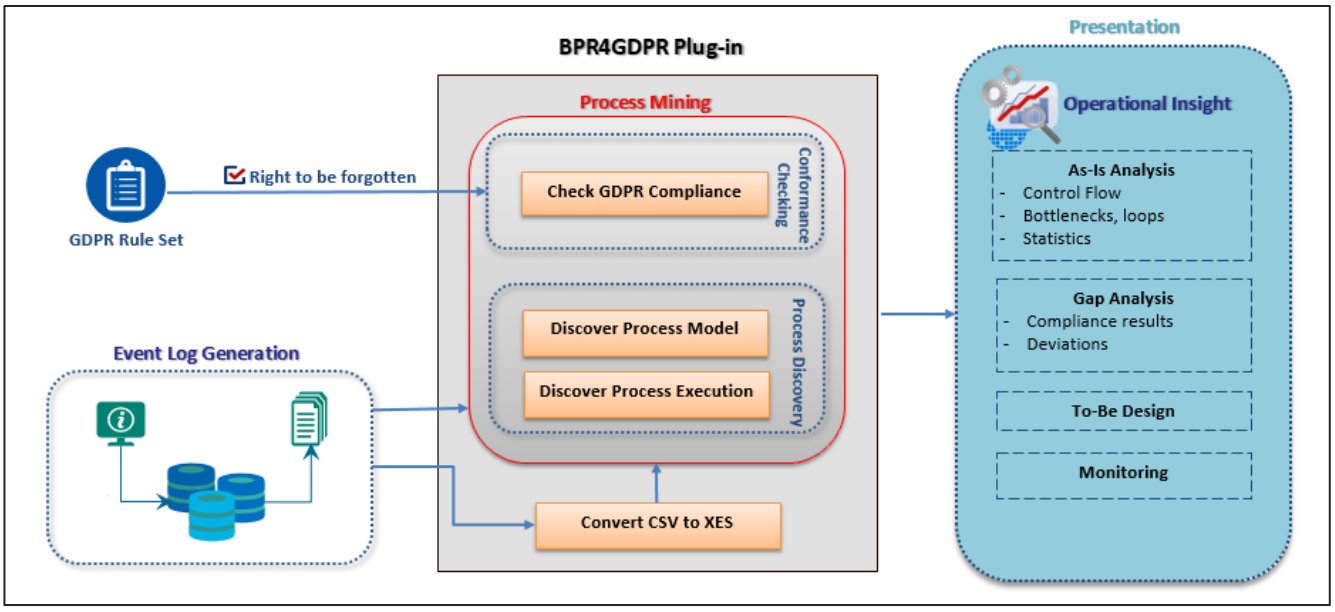


Figure 26: Schematic diagram of the Process Discovery and Mining tool

## 4.3 Specifications of the demo

The car dealership process explained in section 3.1 has been modelled in CPN Tools[5] and simulated for generating a synthetic event log of the process. The process also randomly generates erasure requests during the process simulation. An erasure request stops further execution of the main process and is followed by data deletion activity, subsequently followed by the user acknowledgement/notification activity. Non-compliance to RTBF is injected by adding noise to the event log. The procedure for importing event log to the process discovery and mining tool and the list of the available functionalities are shown in Figures 27-29. The compliance checking configuration is illustrated in Figures 30-31.

For checking GDPR RTBF compliance, *Check the GDPR compliance* plugin requires RTBF in a formal representation in addition to the car dealership process event log. RTBF is provided as a Petri net to the tool in the background. After nomenclature harmonization of the event log and RTBF as in Figure 31, the two entities are confronted to detect and demonstrate deviations/non-compliances in the process execution on case level (cf. Figure 32).

Usually the non-Ω labelled activities are of particular importance from the compliance perspective, though the Ω labelled activities may also contain useful insights. The statistics provided on top of each activity in the visualization consist of two figures: the first figure shows the number of compliant cases while the second figure shows the number of non-compliant cases with respect to that specific activity. Referring to the compliance statistics in Figure 32 of our example demo, out of 264 cases in the event log in 79 cases erasure requests were made by leads. In 9 out of these 79 cases the data subject was not acknowledged about the action taken on the related erasure request.

Along with statistics, the colour filled inside the activities also carry important information. The colour intensity of an activity demonstrates its execution frequency, the darker the colour a particular activity carries the higher relative execution frequency it has. The red coloured perimeter of an activity demonstrates existence of compliance issues with that particular activity. The coloured bar below the statistics also carry information: the Green part shows the relative number of complaint cases with respect to that specific activity, while the Magenta colour shows the relative number of non-compliant cases with respect to that specific activity.

The *Discover process model* plugin discovers the process model on the basis of the provided event log of the real process execution. The output of the plugin for our car dealership use case is provided as Figure 33. The *Discover process execution* plugin discovers a process tree, another member of process modelling family, on the basis of the provided event log of the process execution. Cases in the event log are replayed over the process tree to provide realization of the process execution in run time and also highlight the bottleneck activities. The output of the plugin for our car dealership is provided as Figure 34.

## 4.4 Screenshots and tool download link

Screenshots demonstrating the working of the process discovery and mining tool of the reengineering framework are provided.

[5] http://cpntools.org

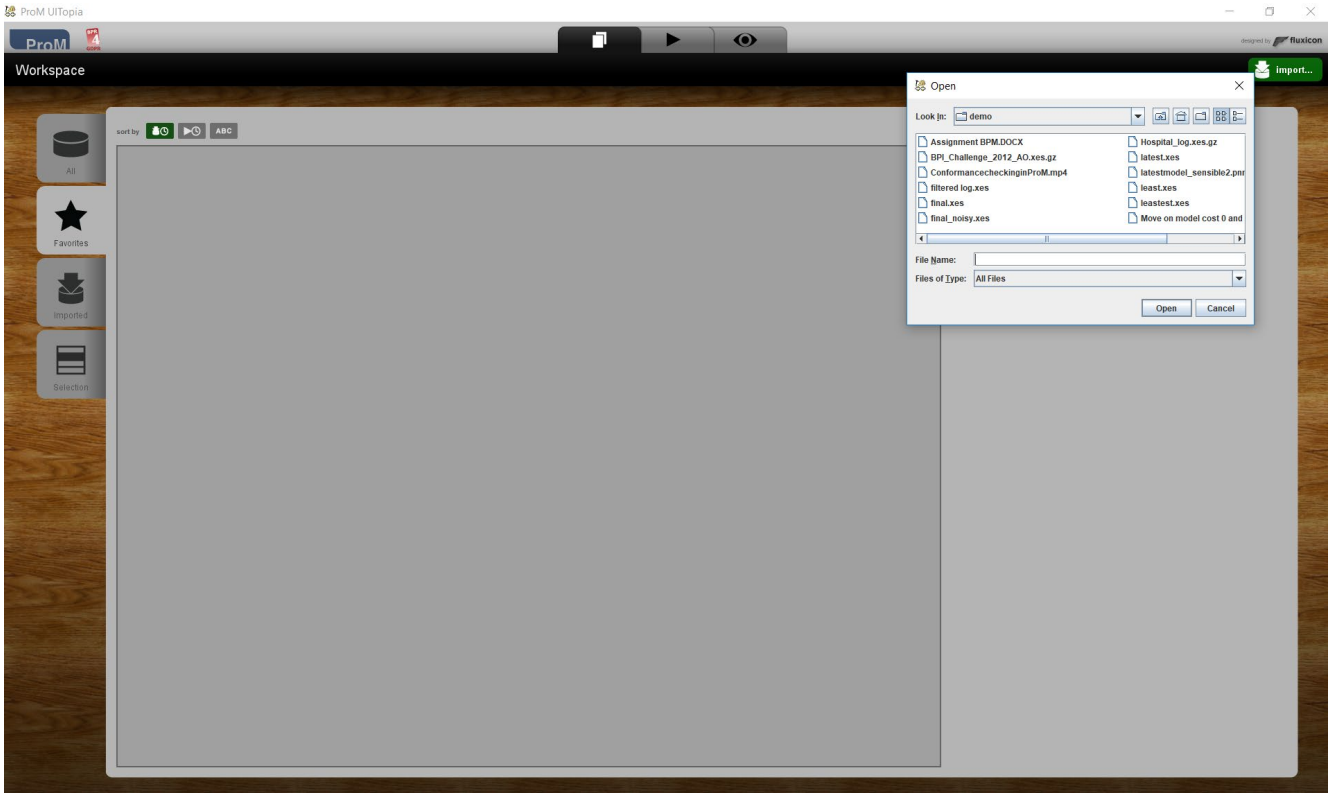


**Figure 27: Importing an event log into BPR4GDPR ProM Workspace**

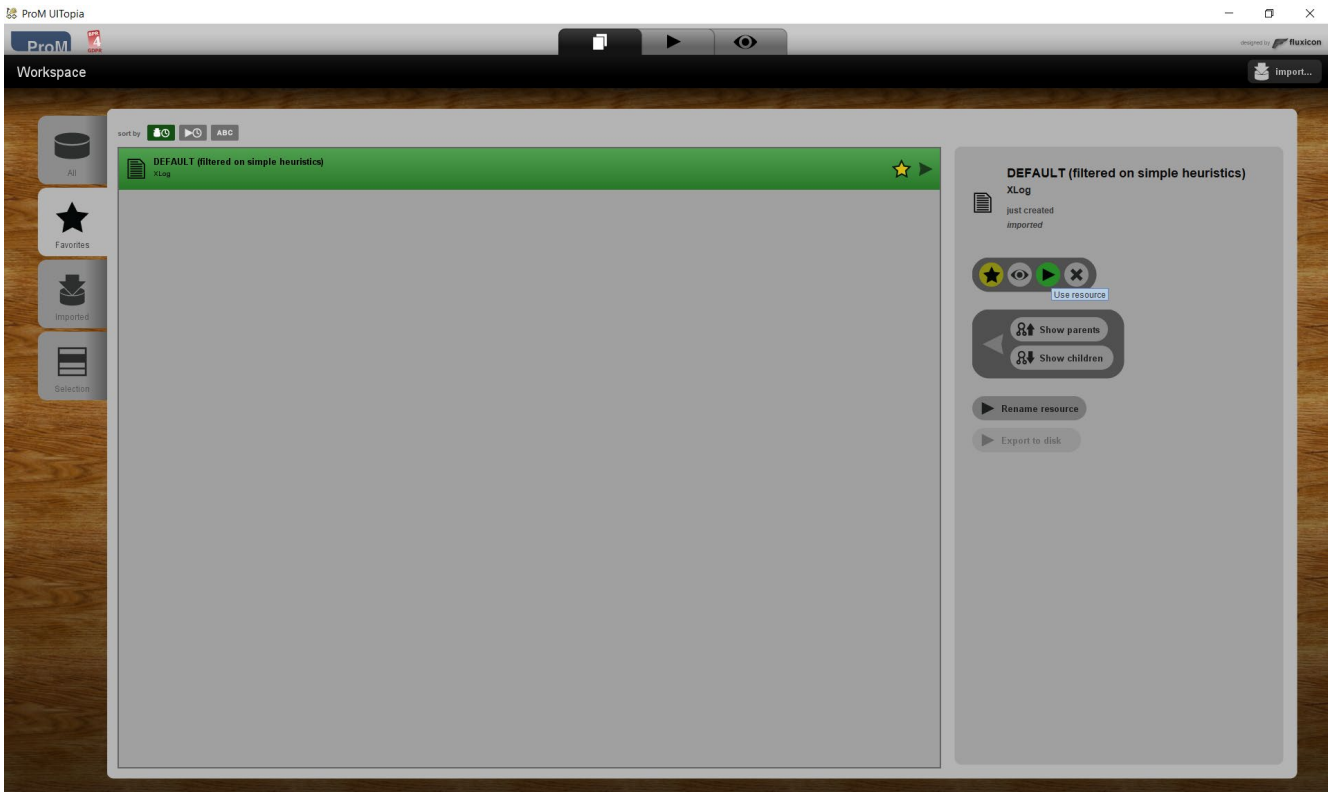


**Figure 28: Displaying the available functionalities on the event log**

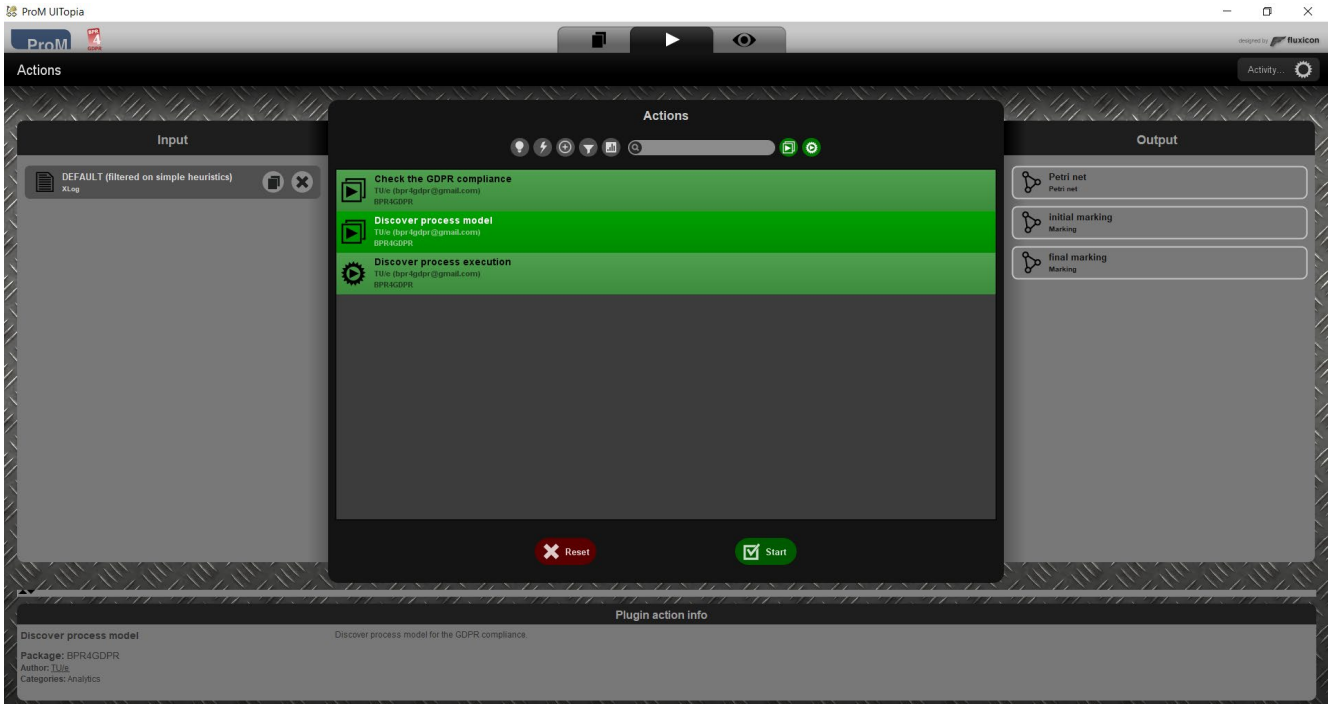


**Figure 29: Choosing the intended functionality on the event log**

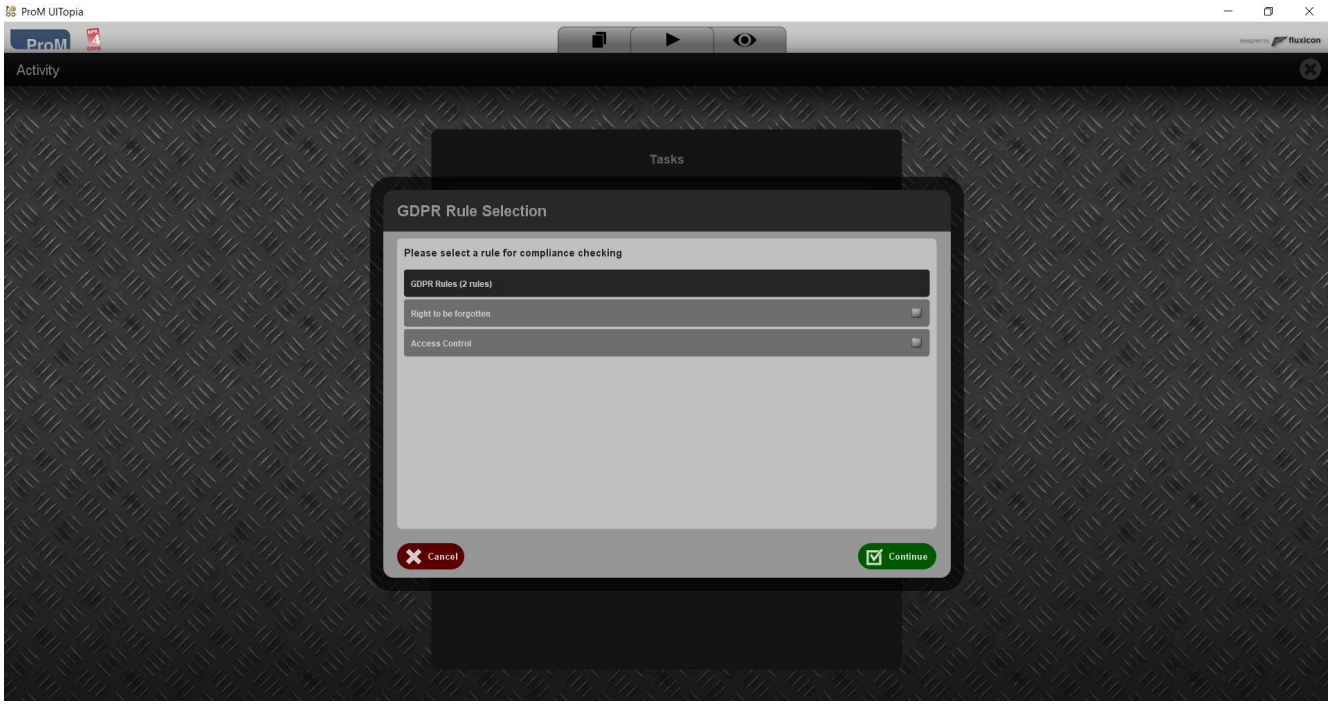


**Figure 30: Choosing the GDPR provision for compliance checking on the event log**

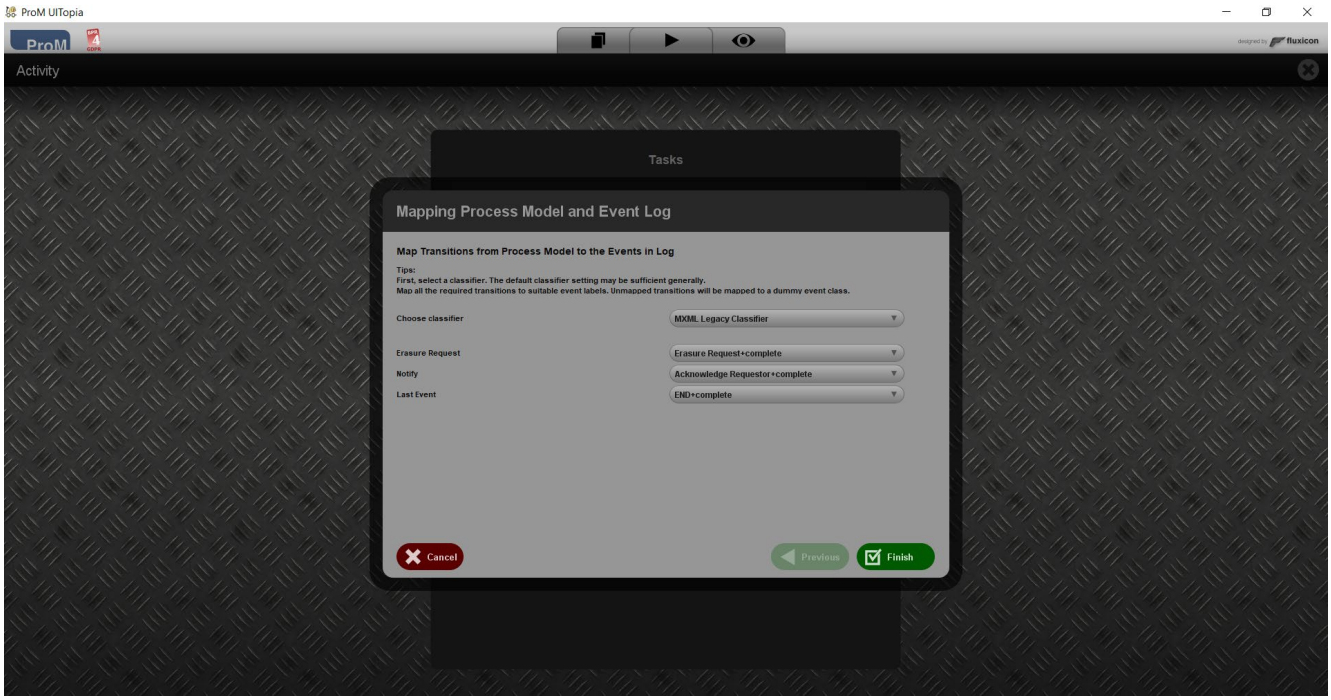


**Figure 31: Mapping the GDPR provision and event log for compliance checking**

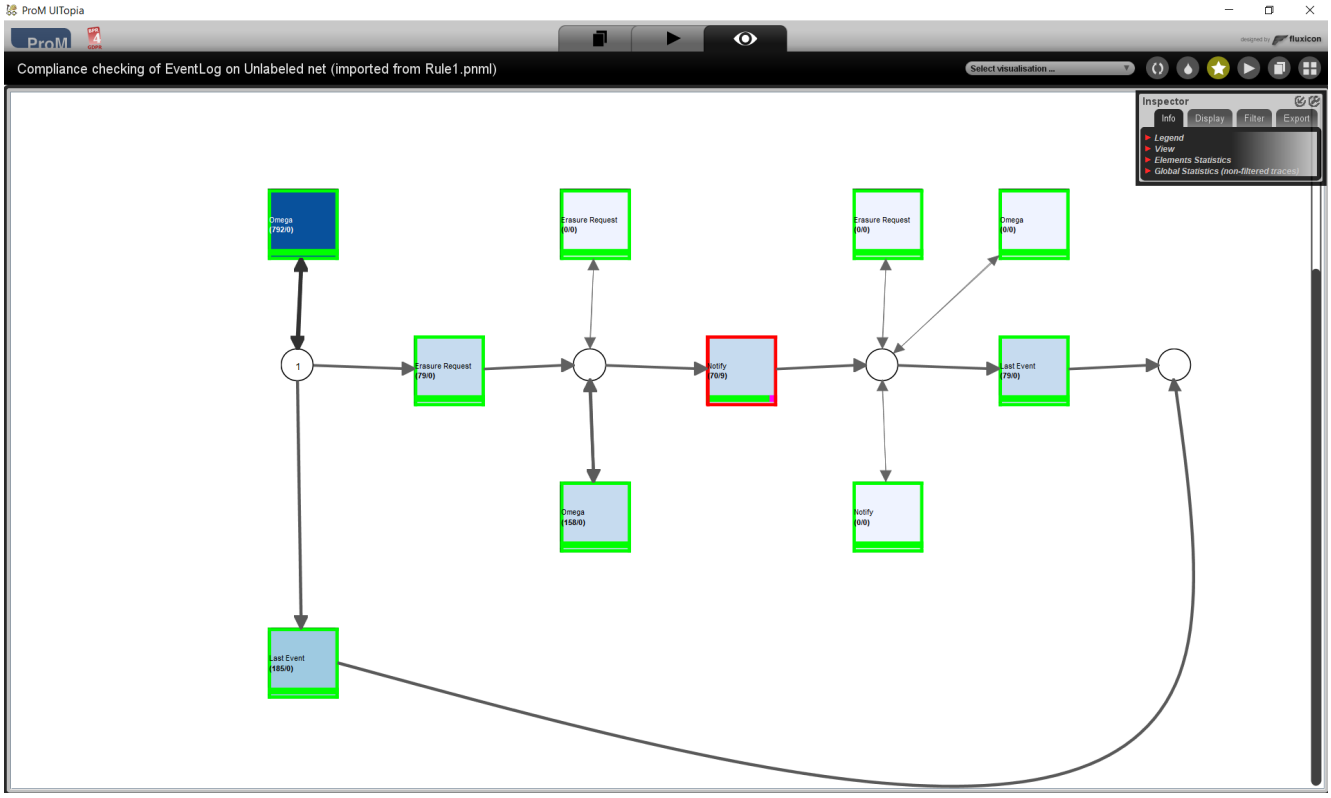


**Figure 32: Compliance checking results**

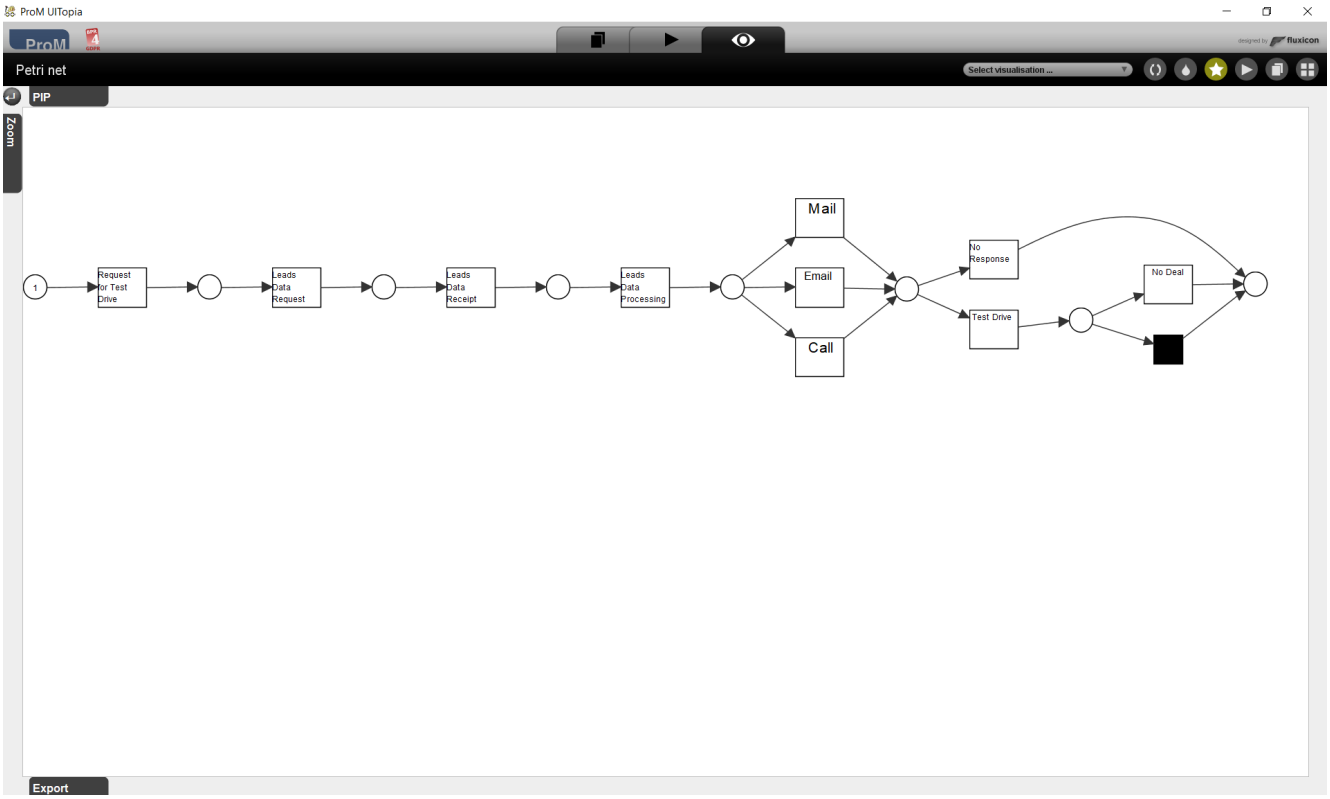


Figure 33: Process model discovery with default settings

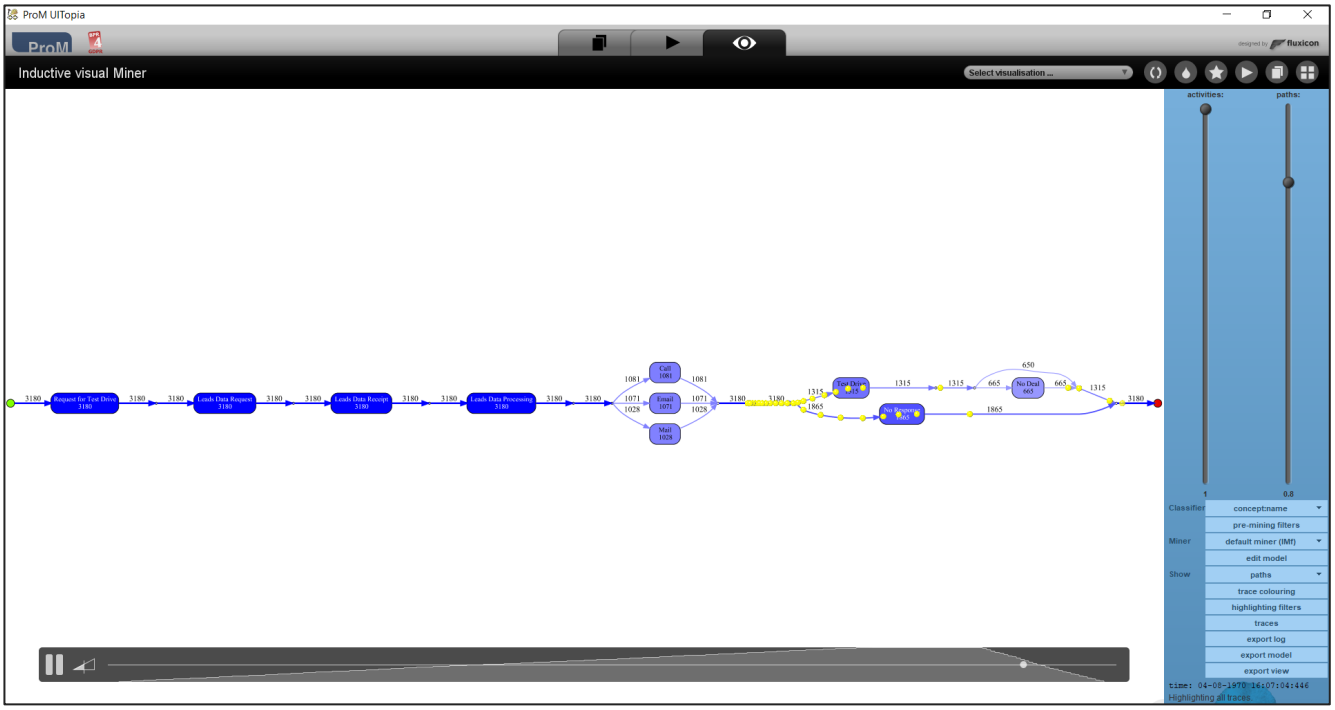


Figure 34: Event Log replay over discovered process tree

### 4.4.1 Download link

The initial process discovery and mining tool is available as Zip archive on the following link. Installation procedure of the prototype is provided as word file with the name “Installing ProM” in the Zip folder. Sample event logs tailored for compliance checking and process discovery are provided for demonstration.

https://surfdrive.surf.nl/files/index.php/s/TRhTu9ZsY6gxAV1

# 5 Future Specifications of the Final Prototype of the Reengineering Framework

Regarding the Planning Environment, the next phase of the WP4 and towards final prototyping of the BPR4GDPR framework will unfold mainly in two directions: the first concerns fine-tuning of verification algorithms and enhanced expressiveness of the Compliance Meta model, in order to support a wider range of GDPR challenges in process-oriented systems; the second will be to promote user experience by incorporating into the user interface more sophisticated process visualisation and specification features.

Also, since every GDPR provision implicates compliance checking on one or more existing process mining perspectives, this poses a challenge with respect to the formalization. Some GDPR provisions might implicate even defining additional process mining perspectives. The envisaged final prototype will therefore cover an extended list of GDPR provisions and their compliance checking mechanisms, by the next phase of the WP4 at M29 of the project.

Actionable information, like process model adaptation, is another envisaged feature of the final process mining and discovery tool of the reengineering framework. The goal is to guide the process stakeholders to adapt the process such that the adapted process on one hand bear maximum resemblance with the primary process but on the other hand be compliant with the GDPR provisions [11], [13]. Visualization is one of the most effective medium for analysts and decision-makers to argue about the process. Improvement in visualization of the compliance diagnostics/statistics and reporting techniques other than visualization which are more comprehendible to the users are to be explored and included as part of the final prototype of the reengineering framework.

# 6 Conclusion

This deliverable provides the initial specification and prototyping of the BPR4GDPR process reengineering framework. The document presented the functionality of the initial versions of the planning environment software and process mining tool.

Planning environment is the software system providing all the necessary functionality for both process modelling and process re-engineering. Regarding the modelling approach, the cornerstone here is the concept of *execution profiles*. First, each comprises by itself an authorisation statement concerning the execution of an operation by some actor(s) on some asset(s), provided that certain conditions hold. Second, they define conditional variants of the task, thus introducing flexibility of authorisations based on the real-time parameters. And third, the elements comprising a profile are defined in great detail following the expressiveness provided by the *entities* primitive, also supported by various features, as are data states, workflow variables, and the explicit consideration of *purpose*. This way, it couples ideas stemming from a wide spectrum of security areas, including SoD/BoD, contextual security policies, as well as attribute-based and privacy-aware access control.

The process discovery and mining tool of the initial prototype has two functionalities. Process discovery functionality discovers the real process in vogue from the execution traces, which can be different from the process perceived to be executing. Compliance checking functionality analyzes process event logs on the ground of the Right to be Forgotten provision of the GDPR and identifies the problematic/non-compliant cases. Non-compliant/compliant cases in the analysed event logs can be filtered in/out for further analysis and targeted remedial actions.

Going a step further, the proposed tools enable the automatic verification of process models, as far as their compliance with privacy principles is concerned, along with their subsequent transformation. This way, it provides for their enhancement with privacy-related features already at design-time, and thus enables inherent privacy awareness.

## References


[1] Cavoukian, A.: Privacy by design: origins, meaning, and prospects for assuring privacy and trust in the information era. In: Yee, G. (ed.) Privacy Protection Measures and Technologies in Business Organizations: Aspects and Standards. IGI Global, Hershey (2012)

[2] Botha, R.A., Eloff, J.H.P.: Separation of duties for access control enforcement in workflow environments. IBM Systems Journal 40(3), 666–682 (2001)

[3] Koukovini, M. N.: Inherent privacy awareness in service-oriented architectures. PhD Thesis, School of Electrical and Computer Engineering, National Technical University of Athens (2014)

[4] Object Management Group (OMG), Business Process Modeling Notation (BPMN) Version 2.0 (2011), https://www.omg.org/spec/BPMN/2.0/

[5] Meda, H.S., Sen, A.K., Bagchi, A.: On detecting data flow errors in workflows. Journal of Data and Information Quality 2(1), 4:1–4:31 (2010)

[6] Kahn, A.B.: Topological sorting of large networks. Communications of the ACM 5(11), 558–562 (1962)

[7] A. Sharp, P. McDermott, Workflow Modelling: Tools for Process Improvement and Applications Development, Second Edition, Boston, Artech House Publishers,2008.

[8] Van der Aalst, W.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Verlag Berlin Heidelberg (2011). https://doi.org/10.1007/978-3-662-49851-4, https://doi.org/10.1007/978-3-662-49851-4

[9] Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. 33(1), 64{95 (Mar 2008). https://doi.org/10.1016/j.is.2007.07.001, http://dx.doi.org/10.1016/j.is.2007.07.001

[10] Carmona, J., van Dongen, B., Solti, A., & Weidlich, M. (2018). Conformance Checking: Relating Processes and Models. Springer.

[11] Zaman, R., and Hassani, M.: Process mining meets GDPR compliance: the right to be forgotten as a use case. 2019 International Conference on Process Mining Doctoral Consortium, ICPM-DC 2019. CEUR-WS. org, 2019. http://ceur-ws.org/Vol-2432/paper6.pdf

[12] Mozafari Mehr, A.S.: Compliance to Data Protection and Purpose Control Using Process Mining Technique. BPM (PhD/Demos) 2019: 108-113 http://ceur-ws.org/Vol-2420/papeDC10.pdf

[13] Hassani, M.: Concept Drift Detection of Event Streams Using an Adaptive Window. 33rd International ECMS Conference on Modelling and Simulation, 2019 June 11 – 14 Napoli, Italy.