Initial specification and prototyping of the policy framework

Deliverable D3.2

Editors

Eugenia Papagiannakopoulou (ABOVO)

Reviewers

Adrián Juan-Verdejo (CAS)

Davide Cascone (BAK)

Date

30th August 2019

Classification

Public

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

1 Introduction

The overall goal of the Work Package 3 is to provide a comprehensive framework for regulating the BPR4GDPR operations by means of security and privacy policies that will reflect the GDPR and related legislation. To this end, the outcome of this Work Package will be a rule-based access and usage control framework, along with the necessary mechanisms for knowledge extraction and decision making, as well as the appropriate information ground, that will be able to drive the compliant execution of operations. This translates to two different missions as regards the role of the policy framework. On the one hand, it is meant to provide the means for system governance in real-time, in the sense that it sets the rules that regulate the operation of BPR4GDPR components. On the other hand, policies comprise the knowledge base that feeds the procedure of process re-engineering, towards compliant by design process models.

This Deliverable reflects the work performed in the context of Task 3.2 “Rule-based access and usage control” and Task 3.3 “Reasoning and knowledge extraction”. In that respect, the BPR4GDPR rule-based access and usage control framework is documented in detail here. Access and usage control in BPR4GDPR aims at handling privacy and security requirements in a holistic manner; access and usage decisions are not taken considering the actions “in isolation”, but taking also into account the operational and data flows representing the interaction between distributed systems. The framework, established upon the Compliance Ontology resulting from Task 3.1 “Compliance ontology”, is rich in semantics and considers all aspects that may affect respective decisions, including types and attributes of all involved entities, contextual parameters, pre-actions and post-actions, events that have occurred. Highly expressive rules at any level of abstraction are specified by means of the Policy Model Ontology providing for complex dependencies among actions and entities, as well as sophisticated separation and binding of duty constraints.

This access and usage control model constitutes the basis for knowledge extraction that, on the one hand, will drive the automated Regulation-aware process (re-)engineering (Task 4.2 “Process verification and transformation”), while on the other, will be leveraged for the evaluation of real-time access and usage requests (Task 5.2 “Data management tools”). Knowledge extraction takes place in two stages. The first phase concerns exhaustive knowledge extraction in an offline manner, including the extraction of meta-rules following hierarchies and relations specified in the Compliance ontology and the subsequent knowledge extraction from the rules set; extracted knowledge through this procedure provides for the assessment of individual access and usage actions, along with conditions that make them valid, i.e., contextual conditions and required and prohibited pre- and post- actions. The second phase, exploiting the results of the first one, refers to real-time decisions regarding the interaction between distributed systems, e.g., associations of tasks in a workflow including the data flow between them; through this procedure, permitted execution scenarios will be derived, involving even modification of the given tasks and flows, along with additional instructions to be followed, in order for the final interaction to be compliant with the underlying policies.

Section 2 begins with the problem statement and a short overview of state-of-the-art access control frameworks and their limitations. After a brief presentation of the Compliance ontology, the basic concept of *action* is introduced, along with the entities involved in the actions’ specification. Access and usage control rules, established upon the concept of action, are presented next. Finally, the Policy Model Ontology leveraged for the implementation of rules is presented.

Section 3 is dedicated to knowledge extraction from the access and usage control rules; this concerns both the afore-mentioned offline procedure and the runtime evaluation of access requests. Of special importance is the

validation and transformation of the so-called *bilateral association*, performed by the policy framework in the context of process verification and re-engineering.

Section 4 presents the logical architecture of the governance component.

Finally, Section 5 documents the status of the associated user interfaces, i.e., for administration of the Information and Policy models.

2 Policy management

Policies are spotlighted at the core of the BPR4GDPR framework, as they constitute the drivers for the compliance-aware process verification and re-engineering, as well as for the run-time operation, providing the behavioural norms of underlying entities. In more detail, privacy and security policies will be incorporated in the processes already during the process specification phase through the Compliance Metamodel or later during the process verification and transformation phase as prescribed by the compliance checks, thus rendering the Regulation-aware re-engineering process the first Policy Enforcement Point (PEP) (to be documented in the Deliverable D4.1 "Initial specification and prototyping of the process re-engineering framework"). On the other hand, run-time policy enforcement will be achieved through the Compliance Toolkit (to be documented in the Deliverable D5.1 "Initial specification and prototyping of the compliance toolkit"), with policies regulating access to and usage of the underlying resources or prescribing the employment of privacy-enhancing mechanisms, such as encryption, anonymisation and pseudonymisation.

In other words, in BPR4GDPR there are two main use cases of enforcement of access and usage control rules that prescribe the requirements for the specification and implementation of the policy framework, as well as for knowledge extraction:

- Enforcement at process specification level, as part of the process verification and transformation procedure offered by the Planning Environment. This transformation is ultimately driven by the goal of privacy compliance and, in this context, the necessary checks and, when needed, the appropriate modifications take place, for the final executable workflow to be compliant. Figure 1 presents an example of process verification and transformation following access and usage control rules.
- Enforcement at run-time, implementing the typical Policy Decision Point (PDP) – Policy Enforcement Point (PEP) interaction. Specifically, PEP functionality is offered by the Data Management middleware that handles data access and usage requests while employing the necessary mechanisms for compliance, like, e.g., data encryption, anonymisation, pseudonymisation, aggregation and user-centric tools expressing data subjects' rights, and controlling data collection, pre- and post-processing, storage and dissemination in a fine-grained way.

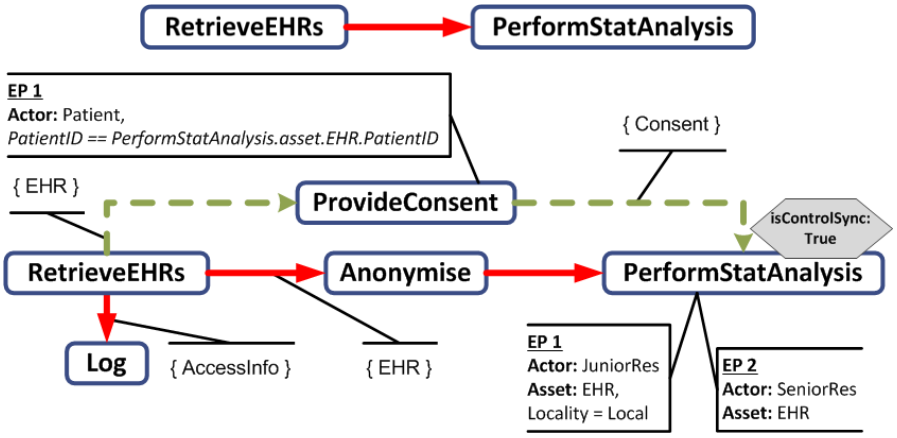

Figure 1: Verification and transformation of task interactions prescribe for process-orientated access and usage control

Traditional access control models, such as the family of Role-Based Access Control (RBAC) [1] models fail to meet the regulatory data protection requirements. In that respect, the development of access control models specifically tailored towards privacy protection has been the focus of intense research in the last years. This

trend, usually referred to as *privacy-aware access control* (cf. e.g., [2]), typically concerns the enhancement of RBAC in order to incorporate different criteria in access control decisions, rather than just *which user*, having *which role*, is performing *which action* on *which data object*; such criteria include *purpose* of data access/usage [3][4], *obligations* denoting complementary actions [5][6][7], *contextual constraints* [8][9][10][11], *Separation and Binding of Duty constraints* (*SoD/BoD*) [12][13] and *attributes* [14][15]. *Attribute-based access control* (*ABAC*) [14] in particular holds the promise of a seismic shift in the development of access control technology tailored for data protection, as stated by Cavoukian et al. in [16]; attribute-based systems provide fine-granularity, high flexibility, rich semantics, among others, features that are more advantageous as organisations move to greater collaboration and data sharing across and outside the enterprise. ABAC is a runtime access control model whereby access to information resources is enforced through the evaluation of policies in an externalised and centralised authorisation solution. Access to functions and data is enforced through contextual attributes that consider the *who*, *what*, *when*, *where*, *how* and *why* of an access request. Closely related to ABAC, Usage Control (UCON) [17][18][19] is considered as the next generation of access control models, handling the problem of authorisation in a continuous way before, during and after access execution, while it also supports mutability of attributes. XACML [20] is the current de facto access control standard implementing ABAC with broad industrial adoption, while Next Generation Access Control (NGAC) [21] is an emerging, relations- and architecture-based standard designed for cloud-based, distributed deployment. NGAC stems from and is in alignment with the Policy Machine [22], a research effort to develop a general-purpose ABAC framework, and one thing that differentiates it from XACML, among others, is that it supports data service-agnostic PEPs [23].

Nevertheless, none of the afore-mentioned approaches completely addresses the requirements for privacy awareness in access control. Moreover, they fail to capture important aspects affecting access decisions in distributed environments, such as the interrelations and dependencies between data collected from different streams originating from heterogeneous sources, the dependencies between loosely-related actions, or the relations among entities of different administrative regimes. In fact, access control must be coordinated across multiple components interoperating for the fulfilment of complex goals, while the automation of security provisions enforcement becomes a critical issue for the effectiveness and consistency of the associated procedures. Besides, the afore-mentioned access control models are not able to control the flow of information between distributed interacting systems, neither during the interaction specification nor during its execution. In this context, work in the area of access control enforcement in workflow management [24][25] and Model-Driven Security [26][27], though important, suffer from enforcing security policies only at run-time and not during the workflow formation.

In light of these issues, BPR4GDPR has adopted a comprehensive access and usage control model, in line with the ABAC implementing standards, allowing for attribute-based data collection and overall handling, and managing all associated constraints, including retention periods and the application of protection measures. It is ultimately grounded upon data protection legislation by means of a Compliance Ontology and thus cover major regulatory aspects, such as support of complementary actions/obligations, consideration of purpose and data subject's preferences, privacy-aware information flow, separation and binding of Duty, context-awareness, and multi-aspect access rights definition [28]. Further, for being effective dealing with complex operations, security and privacy are not enforced solely over single access or disclosure actions. Instead, access and usage control in BPR4GDPR is *process-oriented*; essentially, access and usage control rules comprise the business process compliance ruleset, taking into consideration multiple process perspectives, i.e., control flow, time,

data, resource, and interaction, overcoming the identified verbosity and ambiguity that compliance rules usually present [29] and enabling the process verification and transformation [30].

Overall, BPR4GDPR policy framework allows an authorisation scheme that

- is *declarative*, as it is policy based;
- is *dynamic*, allowing the specification of generic access and usage rules and *run-time decision making*;
- leverages *attributes* of all involved entities (i.e., actor, operation, resource, organisation);
- is *decoupled from the application and data* and is *technology agnostic*, as decisions are made upon semantic types of real-world entities;
- exploits *semantics of relationships* among the involved entities; and
- is p*rocess-oriented*, taking into consideration complex interrelations of actions.

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 thereof.

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 2 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.

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

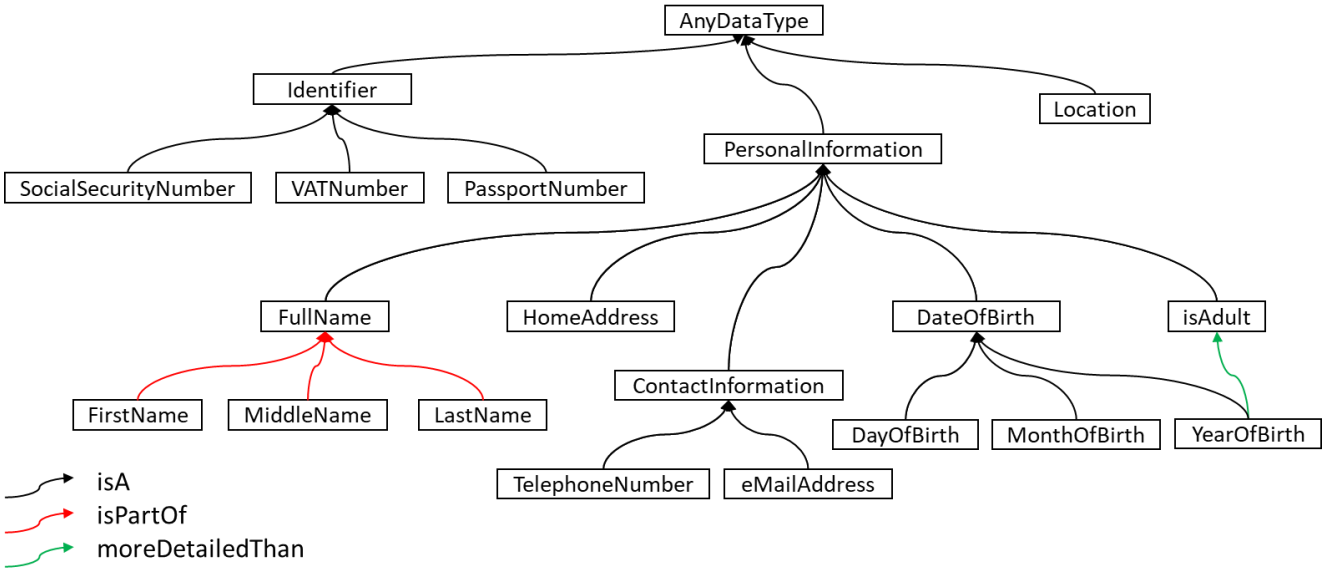

Figure 2: Information Model hierarchy example

2.1.2 Information Model Ontology

Figure 3 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.

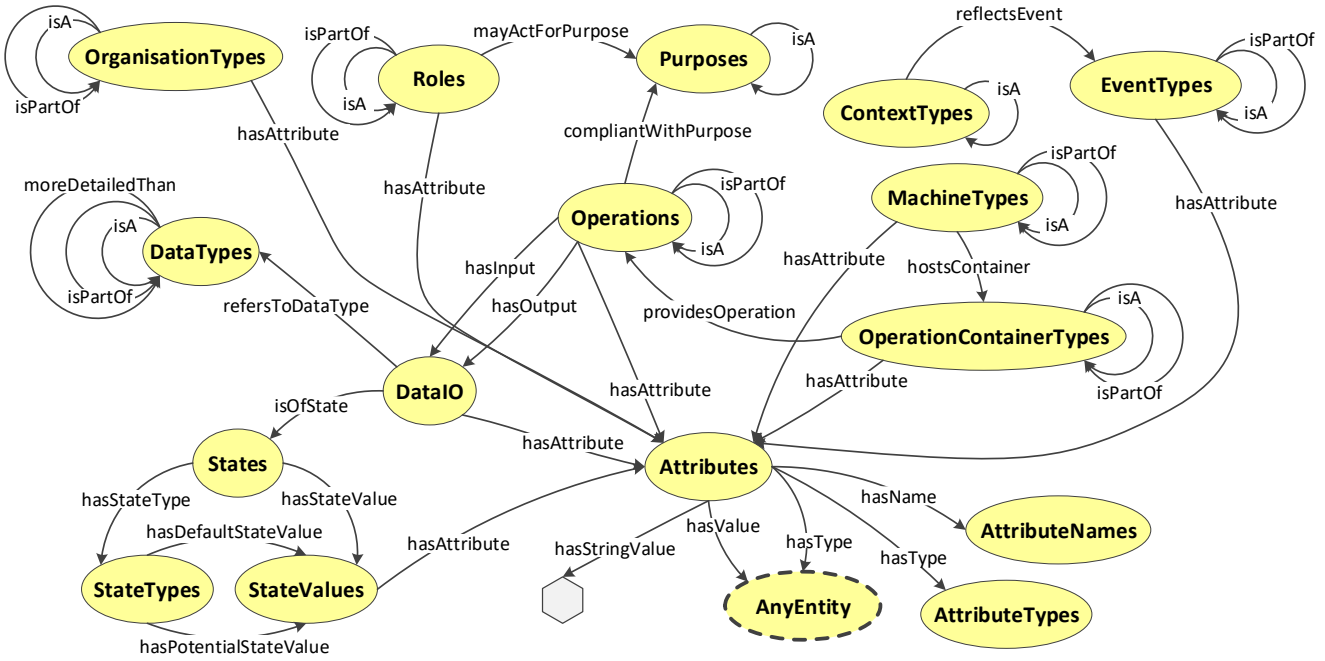

Figure 3: Information Model Ontology

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 BPR4GDPR Policy Framework

2.2.1 Actions

Similar to the *subject—verb—object* linguistic pattern, anything that takes place during the function of a system, can be seen as an *operation* of an *actor* over a *resource*. This applies at any granularity level: at the highest level of a business process, a set of actors performs an aggregated super-operation, consisting of elementary operations, over a set of resources; at a low level, a barcode reader executes a "read" operation over a barcode. Given that a series of such activities, such as a business process, may be executed across different organisations, as in the case of cross-domain processes, the *organisation* within which something takes place comprises an important contextual aspect.

Thus, security and privacy policies are intuitively centred around conceptual quadruples of {*actor*, *operation*, *resource*, *organisation*}; in BPR4GDPR, each quadruple as such is characterised as an *action*, providing the

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

fundamental element for the definition of rules. Actions are formed leveraging the entities of the Information Model; therein, operations and organisations are explicitly defined, whereas different types of entities may play the role of actors and resources, thus be members of the corresponding *Actors* (*A*) and *Resources* (*Res*) sets. Although there are some obvious patterns (e.g., users are typically actors and data are always resources), which entities can be actors and/or resources strongly depends on each organisation, and operational aspects and modelling choices thereof.

More specifically, an action $act \in Act$ is a tuple $\langle a, op, res, org \rangle$, such that: $a \in A$ is an actor; $op \in Op$ is an operation; $res \in Res$ is a resource; and $org \in Org$ is the organisation within which an action takes place.

An action can be either *atomic* or *composite*, depending on whether the associated operation can be decomposed to more elementary operations or not, following the hierarchical relations in *Op*. Actions are also categorised to *abstract*, *concrete* and *semi-abstract*, depending on whether actors and resources are defined at abstract, concrete or mixed level.

Further, it should be stressed that the elements of an action can be specified as *enhanced entities* that include, apart from the entity's semantic type, expressions over its attributes and/or sub-concepts, thus refining the concept definition, towards specifying attribute-based constraints and access and usage control rules. Specifically, an enhanced entity is defined as a tuple $\langle concept, constraint \rangle$, where concept is an IMO element, and constraint is an expression assigning values to concept's attributes and/or sub-concepts.

In this context, two useful mechanisms for defining constraints upon actions' elements, and achieving rich expressiveness in general, are *expressions* and *logical relations*. The latter allow specifying logical structures of concepts/entities leveraging the n-ary operators AND, OR, XOR, and the unary operator NOT. Expressions, on the other hand, enable the definition of constraints on concepts (and contextual conditions, cf. Section 2.2.2); they 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 $exprSubject$. An expression is either an atomic expression or a logical relation thereof.

Specifically, enhanced entities are leveraged in order to express:

- Actor attributes, describing the actor (e.g., a user or a software agent) in an attempt to gain access, characterized by e.g., age, clearance, department, role, job title, etc.
- Resource attributes, describing the object being accessed e.g., data and resources (e.g., medical record, bank account), services (e.g., ePrescription service), and other system components, e.g., a device.
- Operation attributes, complementing the operation being attempted, e.g., the operation *ProvideConsent* can be constrained on the basis of the processing purposes for which the provided consent is valid.
- Environment/contextual attributes, describing attributes relevant to an authorization decision which are independent of a particular actor, resource, or operation; they are generally used to specify the time, location, system status, or other dynamic aspects of the access control scenarios.

Thus, enhanced entities, along with expressions and logical relations, provide the means for expressing constraints like the following:

- *Data subjects* should have access to *their data*, translated into an enhanced entity encapsulating the semantic type A*nyDataType* (*DataTypes* graph) constrained on the basis of data ownership attribute *Owner* (*AttributeNames* set), so that A*nyDataType*.*Owner* = *DataSubject.*
- A physician can view the *prescription record of a patient she has a care relationship with*, which, in a digested format, becomes a user with role *Doctor* can perform operation *ViewPrescriptions* on a resource of type *PrescriptionRecord*, if *PrescriptionRecord*.*Owner*.A*ssignedPhysician* = *Actor*.*userID*; this constitutes in fact an example of a nested constraint (cf. Section 2.2.2)
- A user can edit a *ProjectDeliverable*, *if she is the deliverable editor and it is not yet submitted*, leading to the definition of a complex constraint imposed on a *ProjectDeliverable* comprised of two expressions *logically related* to each other: *ProjectDeliverable.Editor = Actor.userID && ProjectDeliverable.IsSubmitted = false*.

2.2.2 Access and usage control rules

Upon the concept of actions, access and usage control rules are specified; they are defined as *permissions*, *prohibitions* and *obligations* over actions and, since actions can be abstract, concrete or semi-abstract, rules are also specified at these three levels. The format of rules in BPR4GDPR is illustrated in Figure 4; as shown, a BPR4GDPR rule consists of the following elements:

- *action* describes the core of the rule, i.e., what action the rule by definition permits, prohibits or obliges to take place.
- *purpose* reflects the overall objective behind data collection and/or processing; in fact, an operational decision cannot be taken regardless this parameter, which can significantly differentiate an entity's behaviour.
- *pre-action* reflects actions that should have previously taken place in order for the rule to be activated; as an example, there can be the case where a patient should have provided explicit consent (pre-action) in order for her medical record to be processed for research purposes.
- *post-action*[2] similarly implies anything that needs to take place after the enforcement of a rule; for instance, a rule may permit reading some data for providing a service, but may additionally require that the data are deleted immediately after; deny a user access to a document with the obligation to email her manager to let her know that the user tried to access this document; allow Joe to view a document but first watermark the document before returning it to Joe.
- *context* describes conditions defined over "environmental" properties and states, as well as events. For instance, although there may exist a policy allowing doctors access only to medical records of *their* assigned patients, another policy may act as an exception, stating that doctors can view *any* medical record under *emergency conditions*, with the obligation to log the emergency access.

In other terms, an *access and usage control rule* is a structure:

[2] Inline with the XACML3.0 paradigm allowing the definition of obligations and advices, http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

$$\left.\begin{matrix} Permission \\ Prohibition \\ Obligation \end{matrix}\right\} (pu, act, preAct, cont, postAct)$$

where $act \in Act$ is the action that the rule applies to; $pu \in Pu$ is the purpose for which act is permitted/prohibited/obliged to be executed; $cont \in \mathcal{P}(ConT)$ is a structure of contextual parameters; $preAct \in Act$ is a structure of actions that should have preceded; $postAct \in Act$ refers to the action(s) that must be executed following the rule enforcement.

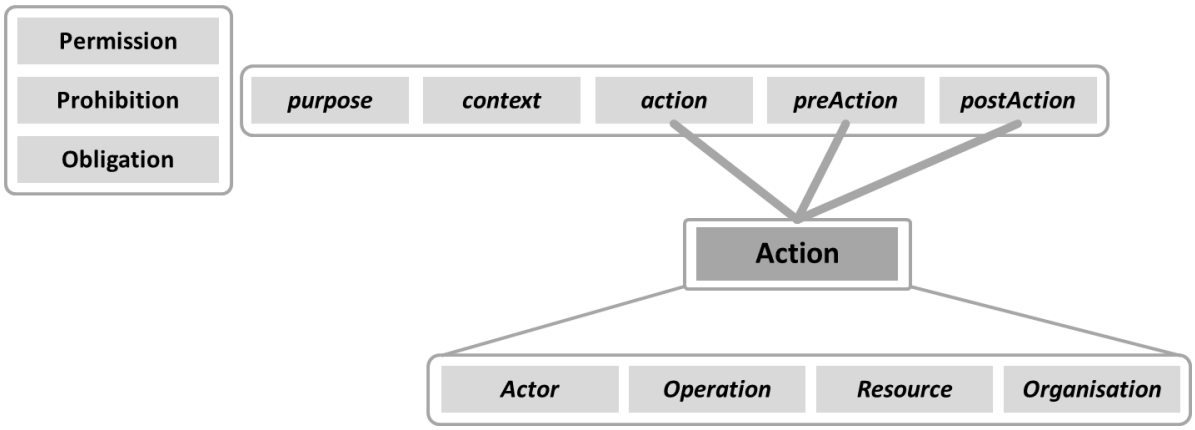

Figure 4: BPR4GDPR rules format

2.2.3 Policy Model Ontology

The rule-based framework described above is implemented as a semantic ontology, referred to as the Policy Model Ontology (PMO), built on top of the Information Model Ontology (Figure 5).

Rules are implemented as instances of the `Permissions`, `Prohibitions` and `Obligations` classes (sub-classes of the `Rules` class), whereas actions are implemented as `Actions` class instances, with $\langle a, op, res, org \rangle$ being reproduced by means of the corresponding object properties. Within an action, the actor, operation, resource and organisation are defined at either the abstract or the concrete level; in that respect, for the representation of an action's elements at the abstract level, instances of the `EnhancedEntities` class are leveraged, constraining the referenced IMO semantic type with respect to its attributes and/or sub-concepts, while for the concrete level, the aforementioned properties point at instances of the class `ConcreteEntities`. Ontologically, constraints are modelled by means of the `Expressions` and `LogicalRelations` classes; instances of the former essentially model atomic constraints, whereas the latter, implied by thick lines in Figure 5, provide for structuring composite expressions out of atomic ones. Appropriate properties are defined for `Expressions` individuals, indicating the subject (`hasExprSubject`), operator (`hasOperator`) and value (`hasExprValue`).

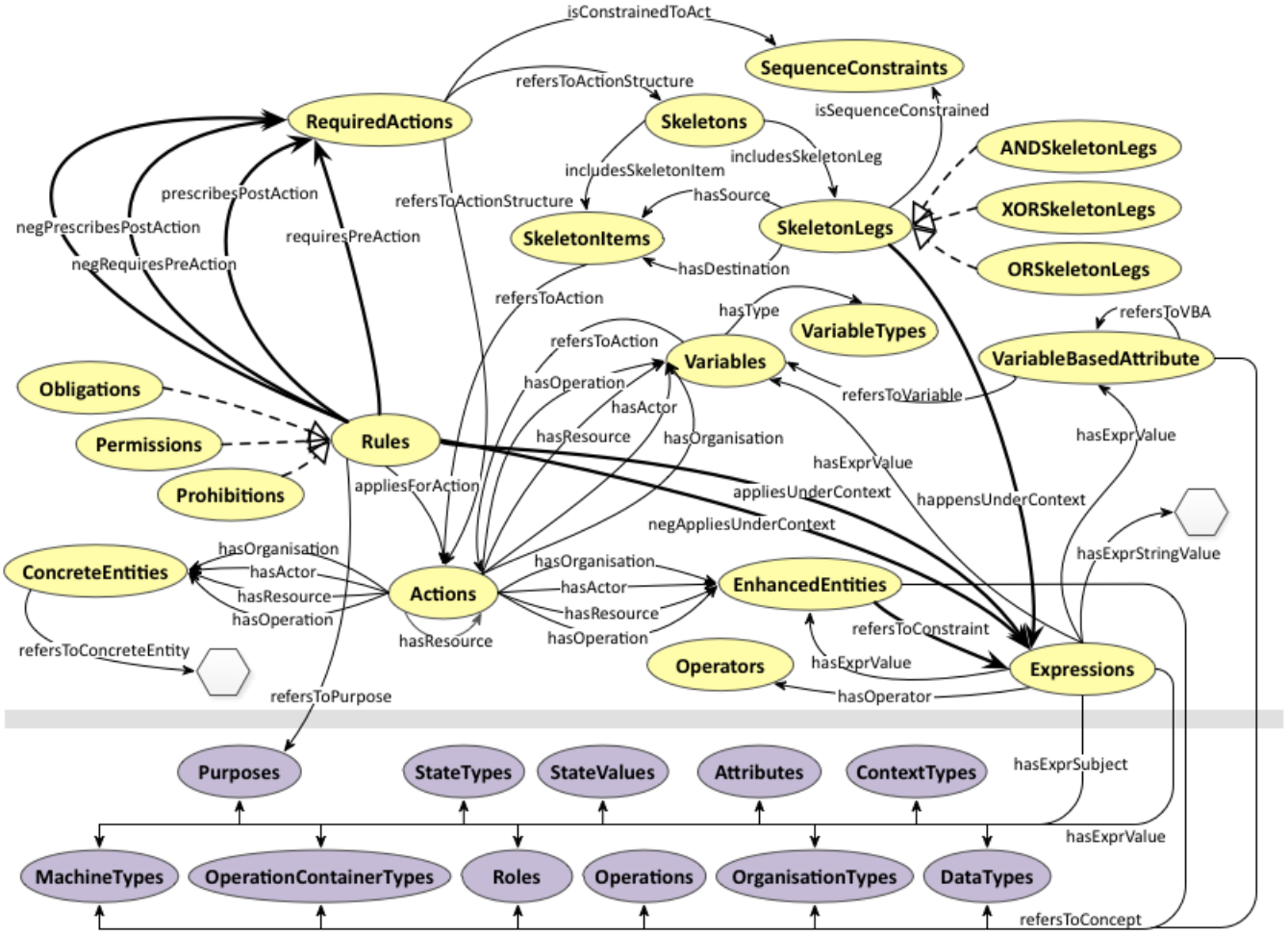

Figure 5: Policy Model Ontology (PMO)

As constraints may vary from quite simple to very sophisticated ones, constituting the cornerstone of BPR4GDPR's access control expressiveness, various data structures are leveraged for covering all the cases of possible expression values; in that respect, the value of an expression can be:

- a String expression representing the most elementary case (use of the datatype property `hasExprStringValue`, e.g., for constraining an actor on the basis of her name)
- an IMO instance, e.g., for denoting that the processing purposes for which a patient provides consent include `ePrescription` (instance of the `Purposes` class)
- a `Variables` instance; variables are necessary for pointing to other actions' or rule's elements, thus serving as placeholders in the context of a rule, i.e., the actor, operation, resource, organisation of an action, a whole other action or the rule's purpose; in that respect, a variable has a type (a `VariablesTypes` instance, e.g., `Actor`, `Purpose`) and points to an action (`refersToAction` object property), when this is necessary, e.g., for denoting the actor of some pre-action.
- an `EnhancedEntities` instance, for modelling nested constraints like the "*PrescriptionRecord.Owner.AssignedPhysician = Actor*" one; the entity `PrescriptionRecord` has to be constrained upon its `Owner` attribute (first `EnhancedEntities` instance) and the owner, being

a Patient, will in turn be constrained upon its AssignedPhysician attribute (second nested EnhancedEntities instance assigned as the value of the first one's constraint, see also Figure 7).

- A VariableBasedAttributes instance, 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.

It should be noted that apart from EnhancedEntities instances, all action elements may be specified by means of Variables and VariableBasedAttributes instances.

Actions instances are used for specifying the main action and pre- and post-actions of a rule; in the latter cases this association is indirect, with the RequiredActions class mediating and enabling the specification of time and sequence constraints. Beyond such constraints, the BPR4GDPR approach incorporates a mechanism for combining actions, so as to form complex structures thereof, referred to as *skeletons*, following various sequence patterns. Rules instances are also associated with the corresponding purpose and contextual conditions. In this context, refersToPurpose property maps a rule with a (IMO) Purposes instance, whereas appliesUnderContext and its negative equivalent negAppliesUnderContext point at an Expressions or LogicalRelations instance, declaring the contextual parameters under which the considered rule applies. Finally, dependencies among all the entities comprising the actions of a rule enable the specification of advanced SoD and BoD constraints, instead of relying only on role-/user- centric constraints.

Figure 6 and Figure 7 depict the ontological implementation of two rules declaring that "data subjects should have access to their data"[3] and that "doctors can view prescription records of their assigned patients", respectively. The first rule is implemented as a generic (high-level of abstraction) permission and will be accordingly propagated across the corresponding graphs by means of the meta-rules extraction mechanism (cf. Section 3.1), thus, the majority of the rule's elements are defined by means of IMO graphs roots (AnyPurpose, AnyOperation, AnyDatatype), while a Variables instance is defined for associating the actor of the access action (being a DataSubject) to the requested resource. The rule illustrated in Figure 7 provides an example of the afore-mentioned nested constraints. Figure 8 depicts a more complex rule, which requires as a precondition for providing access the execution of another action (the provision of data subject's consent) and prescribes the execution of a third action (logging of access) at the same time access is permitted; in fact, this rule constitutes the implementation of default rule No8 of deliverable D3.1 "No data processing should take place unless the data subject has provided consent", with the addition of the logging obligation. Of special interest is the extended use of variables in order to point to the rule's purpose (Var#3) and the access action itself (Var#2), while for the definition of the pre-action's actor a VariableBasedAttributes instance is leveraged (VBA#1), thus associating Act#4 with the resource of Act#2 (Var#4) and specifically with its attribute Owner (implementation of $Actor_{preAction} = Resource_{action}.owner$ constraint). Finally, it should be noted that no actor and operation are defined for the access action (meaning that the system has to deduce that the rule will apply for all possible types of access requestors, being them human users or software agents, intending to perform any type of operation on the requested resource)[4].

[3] Implementation of default rule No7 of deliverable D3.1 — Compliance Ontology

[4] Organisation definition is omitted in the depicted examples for simplicity reasons; AnyOrganisation is implied.

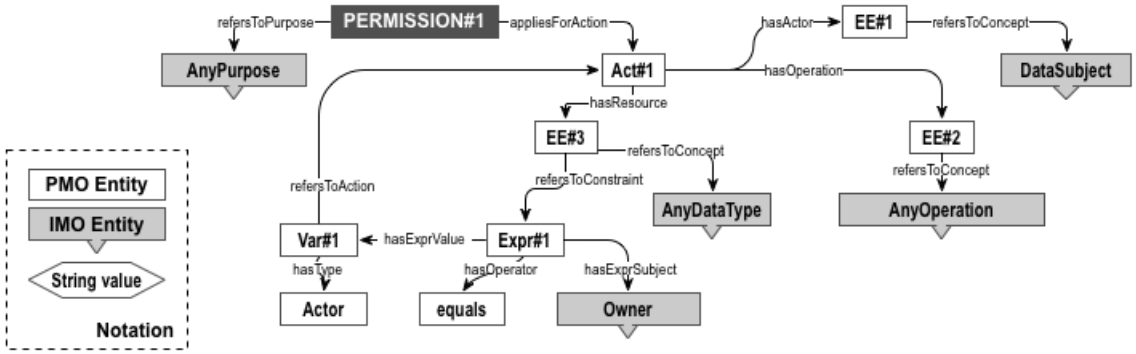

Figure 6: Data subjects should have access to their data

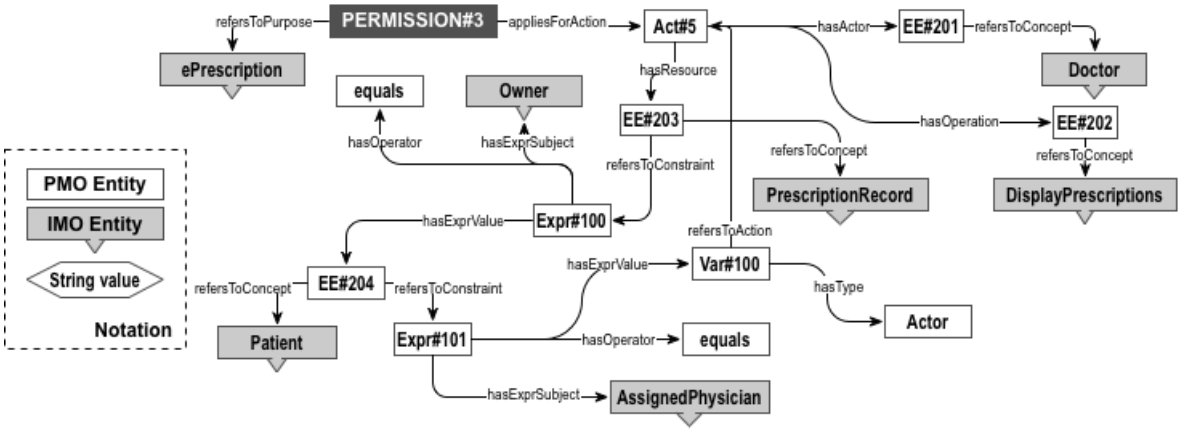

Figure 7: Doctors may read prescription records of their own patients for the purpose of ePrescription

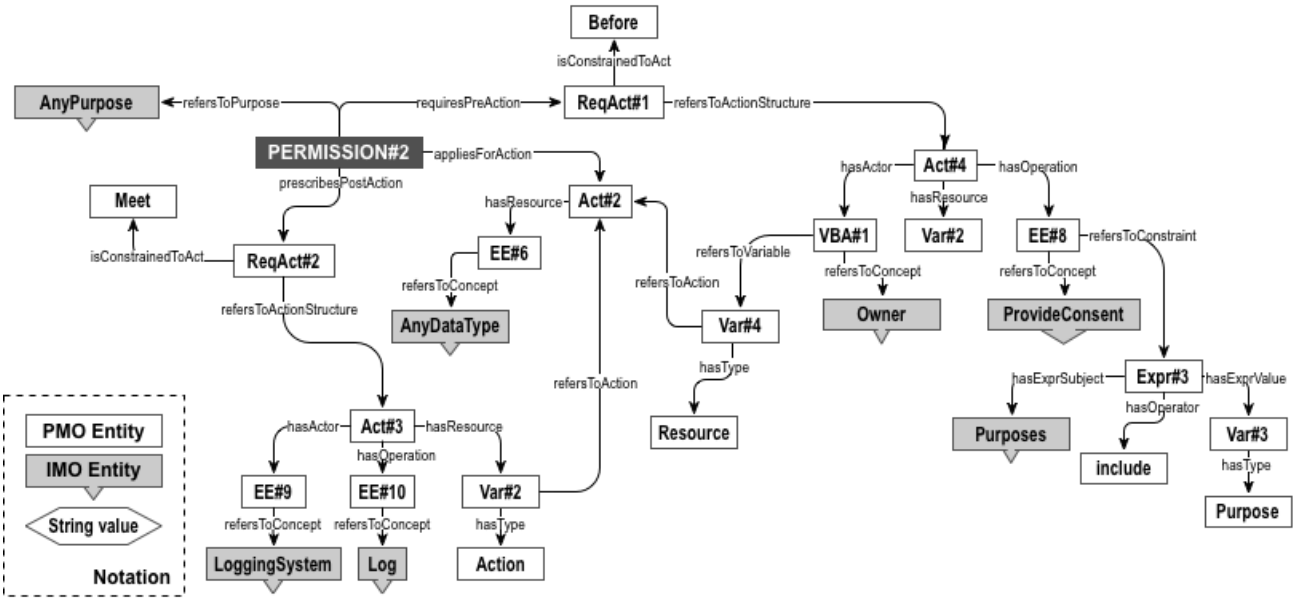

Figure 8: Data processing is allowed if the data subject has provided consent for this processing and the specific processing purpose; additionally, access will be immediately logged

3 Knowledge extraction

The afore described ontologies constitute the Knowledge Base of the system, allowing for inference of knowledge related to the access to the system resources. However, the inference procedure is complex and can be quite time consuming, considering the expressiveness —and the complexity introduced by the latter— of the underlying ontologies. Therefore, the maximum possible information should be made available already before it will be requested in the context of a bilateral association's verification procedure or before a real-time access request submitted by the Data Management middleware. In other words, all the heavy processing tasks are performed offline and only when the ontologies are updated, for instance, when new access control rules are added or existing ones are revoked, thus achieving performance gains.

3.1 Offline Knowledge Extraction

The Offline Knowledge Extraction consists of two procedures; the *extraction of meta-rules*, following inheritance patterns based on the properties of the Semantic Information Model, and the *reasoning procedure on the basis of the access and usage control rules*, which, through rules analysis, combination and conflict resolution, results in the extraction of all the actions that are permitted to be performed in the context of the system's operation, along with conditions that make them valid, i.e., a valid purpose, contextual conditions and required and prohibited pre- and post- actions structures.

3.1.1 Extraction of Meta-rules

As the Access and Usage Control Model considers various hierarchies, rules defined at a high level of abstraction are propagated across the corresponding IMO graphs. Thus, starting from the explicitly specified rules, initially comprising the *Rules* (*RU*) set, meta-rules are generated, following specific inheritance patterns. The latter are based on the `isA`, `isPartOf` and `moreDetailedThan` IMO properties, while they depend on the type of the rule, that is whether it is a permission, a prohibition, or an obligation. Indicatively, for two IMO concepts con_i and con_j:

- $isA(con_i, con_j) \rightarrow inheritsFrom(con_i, con_j)$, for all types of rules
- $isPartOf(con_i, con_j) \rightarrow inheritsFrom(con_i, con_j)$, for prohibitions (inversely for positive authorisations)

The afore mentioned patterns apply to the inheritance of rules with respect to the purpose and the entities of the access action. Regarding pre- and post- actions, no meta-rules are generated according to the inheritance patterns concerning the involved action entities. Instead, pre- and post- actions are *unfolded*; new actions are constructed by substituting the initial pre-/post- action's elements with the corresponding most specific ones (leaves of the *isA* graphs). This way, each action participating in the original pre-/post- action structure leads to a set of *equivalent* actions, which are alternative to each other, meaning that any related pre- or post- condition will be satisfied through execution of any of these equivalent actions.

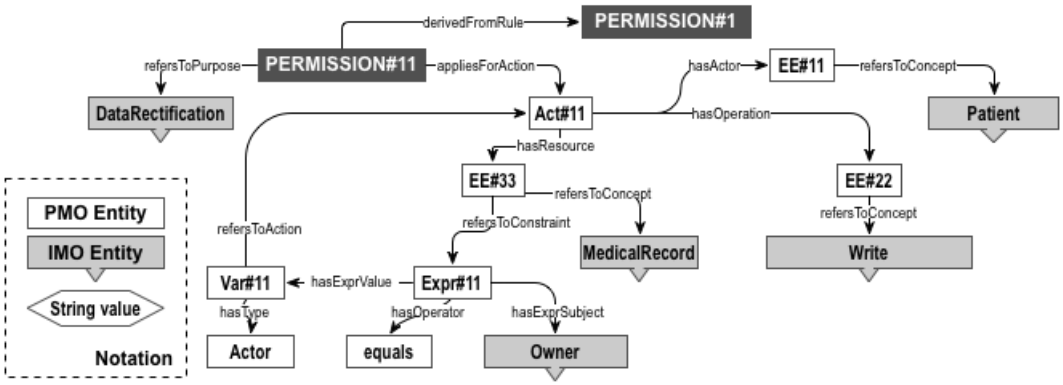

Figure 9: Meta-rule derived from rule of Figure 6

Figure 9 is a meta-rule derived from the permission of Figure 6; a `Patient` should have `Write` rights to *her* `MedicalRecord`, for the purpose of `DataRectification`, derived because of the following relations[5]:

- $isA(DataRectification, AnyPurpose) \rightarrow inheritsFrom(DataRectification, AnyPurpose)$, met in the *Purposes* graph
- $isA(Patient, DataSubject) \rightarrow inheritsFrom(Patient, DataSubject)$, met in the *Roles* graph
- $isA(MedicalRecord, AnyDataType) \rightarrow inheritsFrom(MedicalRecord, AnyDataType)$, met in the *DataTypes* graph
- $isA(Write, AnyOperation) \rightarrow inheritsFrom(Write, AnyOperation)$, met in the *Operations* graph

Regarding the permission of Figure 8, at first, `ReqAct#1` is unfolded based on the specialisation relation `isA` of the operations graph

- $isA(ProvideEConsent, ProvideConsent) \rightarrow inheritsFrom(ProvideEConsent, ProvideConsent)$, and
- $isA(ProvideOralConsent, ProvideConsent) \rightarrow inheritsFrom(ProvideOralConsent, ProvideConsent)$,

so that finally it is replaced by two alternative pre-actions, as the `LogOR#1` instance implies. Subsequently, meta-rules are extracted based on the inheritance patterns holding for the purpose and the access action's entities. Missing entities, actor and operation in the case of the permission of Figure 8, are added based on the relations

- $isA(Doctor, AnyRole) \rightarrow inheritsFrom(Doctor, AnyRole)$, met in the *Roles* graph and
- $isA(DisplayPrescriptions, AnyOperation) \rightarrow inheritsFrom(DisplayPrescriptions, AnyOperation)$, met in the *Operations* graph.

Each meta-rule, for rules' administration reasons, is annotated with information denoting the (meta-)rule from which it was derived through the object property `derivedFromRule`. Finally, it should be noted

[5] It is noted that Permission#11 will not be in fact directly derived from Permission#1; meta-rules are generated in multiple steps, each time replacing just one element with e.g., a more specific one. However, we present the permission with all the high-level concepts replaced by the most specific ones for simplicity reasons.

that variables and variable-based attributes are not replaced or unfolded at any stage of offline reasoning, so that association to the referred entities is not lost until the very time of requests' evaluation.

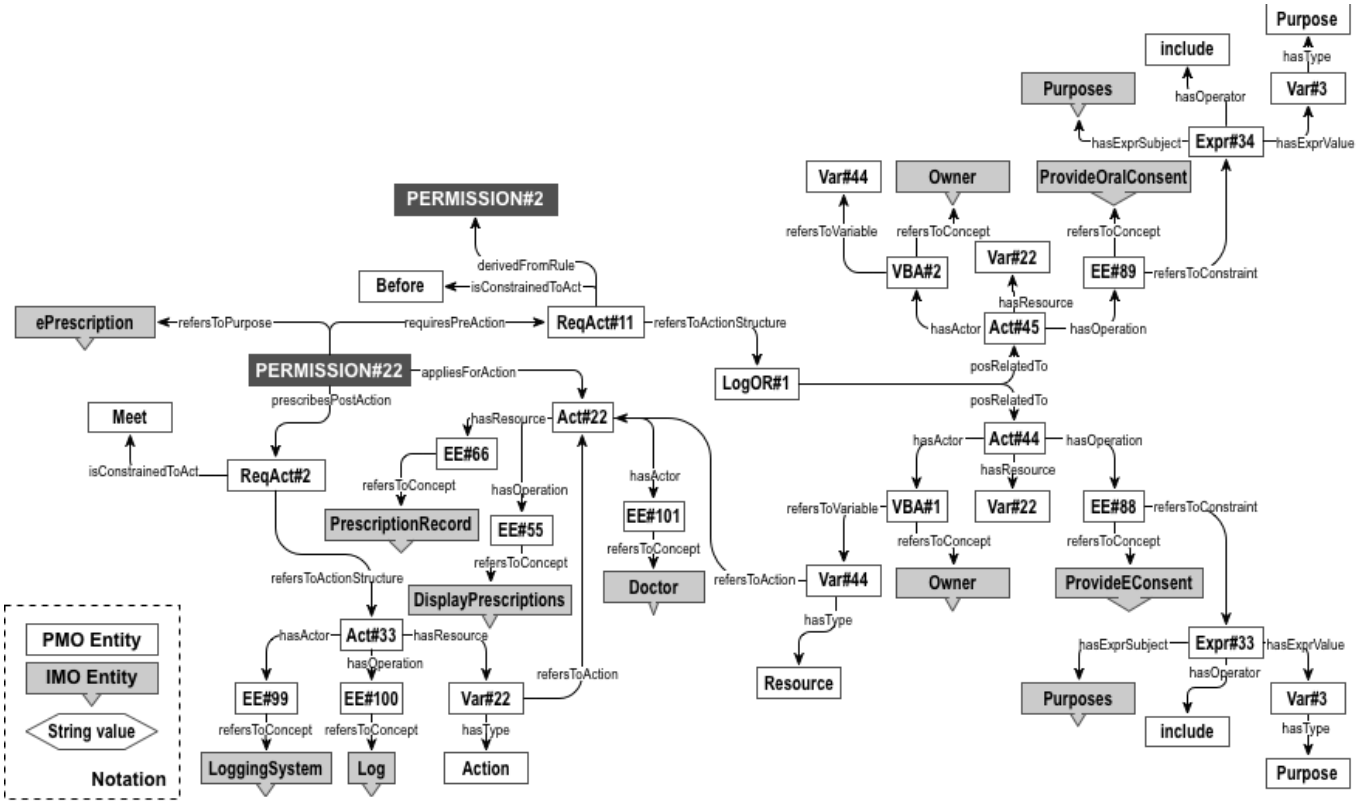

Figure 10: Meta-rule derived from rule of Figure 8

3.1.2 Reasoning upon Access and Usage Control Rules

The core PMO described so far is extended in order to support the proactive extraction of knowledge contained in the access and usage control rules. In this context, and as illustrated in Figure 11, two classes have been specified, namely `PermittedActions` and `OfflineRequiredActionStructures`, allowing the combination of knowledge contained in different rules.

Instances of the first class represent all the actions which are in principle permitted to be executed in the context of the system's operation (possibly under certain conditions). Specifically, these actions are *totally specialised*, meaning that the involved action entities refer to IMO *leaf-elements* of the respective *isA* graphs (unless they constitute concrete entities). This choice simplifies the bilateral association verification procedure, where each original task should be replaced by the most specialised equivalent one in order to reduce the real time requests to the underlying ontologies, while the fact that a permission may be defined at a high level of abstraction, with exceptions defined along the corresponding graphs, was taken into consideration as well; in fact, this procedure encapsulates a conflict resolution process, combining knowledge from different rules referring to the same access action. `PermittedActions` instances are derived by the specified `Permissions` instances, as for each in principle valid action there should be at least one permission; the said action is mapped to the access action of the reference permission. In essence, a permitted action encapsulates all the information contained in a rule, that is, the purpose and the contextual conditions under which it is valid, as well as the required or forbidden action structures complementing its execution, while grouping of permissions in a single `PermittedActions` instance is also possible. Therefore, the corresponding properties are specified in the

extended ontology, i.e., `refersToOriginalAction`, `validForPurpose`, `validUnderContext` (and the corresponding negative property), `requiresActionStructure` and `forbidsActionStructure`.

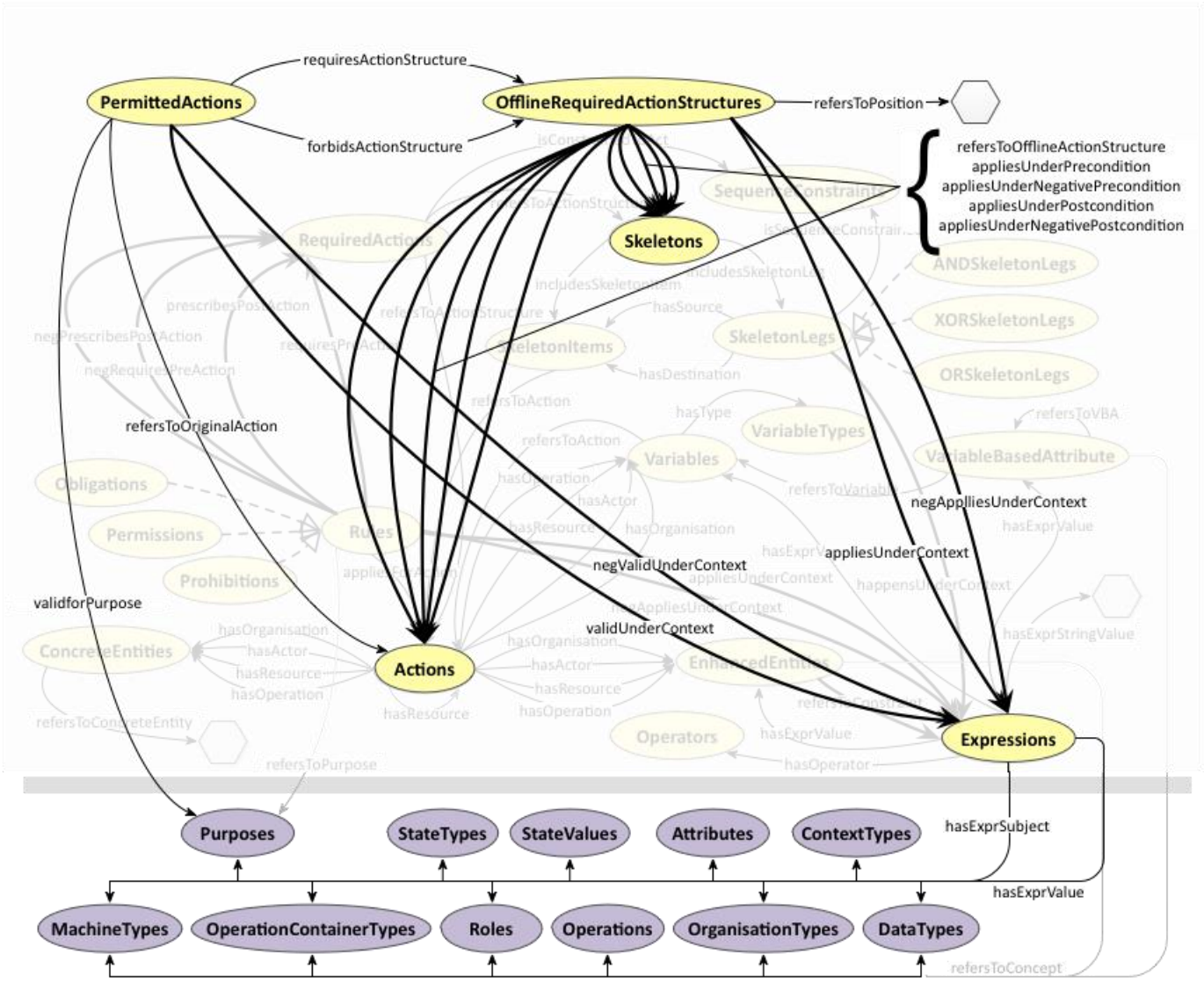

Figure 11: PMO Extension

On the other hand, `OfflineRequiredActionStructures` class reflects the required or forbidden pre- and post-actions complementing the considered permitted access action. In accordance with the `RequiredActions` class of basic PMO, instances of this class refer to either single actions or skeletons (or logical expressions thereof), while they appoint the position of the underlying action/s with respect to the reference permitted action. Additionally, preconditions and postconditions may be defined for an offline required action structure (by means of the `appliesUnderPrecondition` and `appliesUnderPostcondition` properties) for maintaining the interrelations between the access action and the pre-/post- actions of a rule, as well as positive and negative contextual conditions. As opposed to the `PermittedActions`, instances of this class are derived not only from `Permissions`, but from any kind of rules. Essentially, while constructing a permitted action, for each original access action all the permissions, prohibitions and obligations by which this action is referred, either as the main action of the rule or as a pre-

/post-action, are gathered. In the case of a permission, the specified pre- and post-actions are directly mapped to offline required action structures, with the permitted action referring to the main action of the rule, while in the case of prohibitions and obligations, we search for those where the considered permitted action participates in the pre-action structure of the rule, as the execution of the reference action will activate the respective prohibition/obligation.

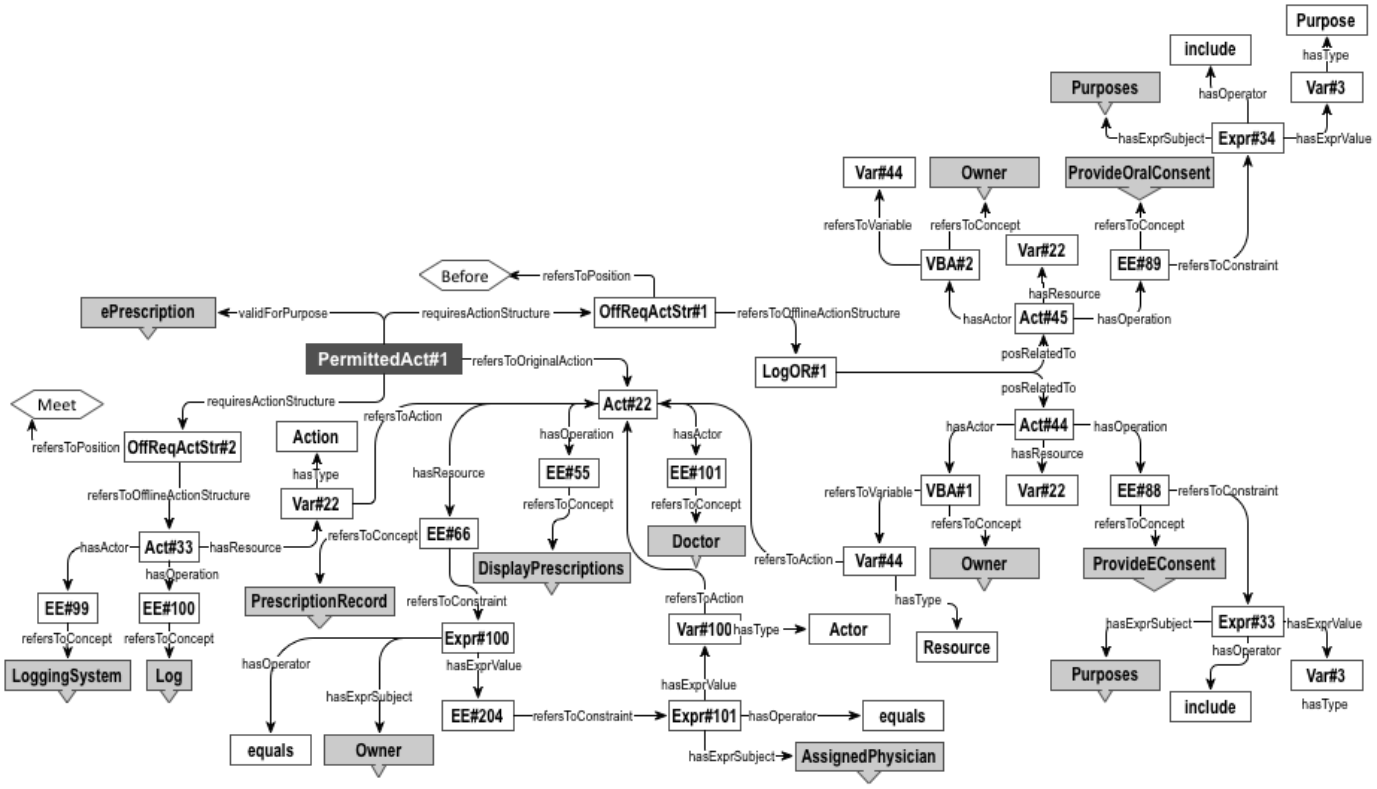

Figure 12: a `PermittedActions` instance

Figure 12 presents a permitted action extracted through combination of the permissions of Figure 7 and Figure 10; although both permissions apply for the action ⟨`Doctor`, `DisplayPrescriptions`, `PrescriptionRecord`⟩, `PermittedAct#1` is actually established upon `Permission#3`, as the latter further restricts the requested resource upon its ownership attribute, whereas the offline required action structures `OffReqActStr#1` and `OffReqActStr#2` are specified as well, according to the pre- and post-actions of `Permission#22` (`ReqAct#1` and `ReqAct#2`).

Summarising, these two classes essentially provide the necessary knowledge for answering the following questions, which are very critical in the context of the verification procedure of a bilateral association:

- Whether an action is in principle valid, i.e., the actor *a* is permitted to perform the operation *op* on a resource *res* within an organisation *org*; in principle valid means that an action may be valid for some purpose, however it may turn out invalid due to, e.g., some contextual constraints or conflicts with other actions.
- Whether an action requires the presence of one or more other actions, along with their absolute or relative position with respect to the action under question, taking into consideration various factors, like the underlying purpose, context, etc.

- Whether inconsistencies and conflicts exist between two actions, taking into consideration various factors, like the underlying purpose, context, etc.

3.2 Evaluation of requests

As afore-mentioned, in BPR4GDPR there are two main categories of requests that the PDP receives and evaluates. On the one hand, reasoning over the access and usage control rules feeds the procedure of process verification and re-engineering, towards compliant by design process models; thus, the first category of requests comprises the ones originating from the Planning Environment and centres around the compliance of given tasks interactions at workflow specification level. On the other hand, the Data Management middleware acts as Policy Enforcement Point (PEP) enforcing the fine-grained access and usage control; for achieving these purposes it utilises knowledge extracted through reasoning, policy decisions as well as privacy-enhancing mechanisms and user-centric tools expressing data subjects' rights. In this context, requests from the Data Management middleware emerge at the time access to a resource is attempted, i.e., at process execution phase.

3.2.1 Bilateral Association Verification

The verification procedure offered by the Planning Environment is to a great extent based on the compliance requirements prescribed by the PDP. For this purpose, starting from a workflow model, the Planning Environment passes to the PDP three structures through the Compliance Interface (cf. Figure 21), containing respectively: the *declared purposes* that the workflow should serve; the *initiators* intended to trigger its execution for each of these purposes; *all bilateral task associations contained in the workflow*, i.e., all the pairs of all interacting tasks in the workflow, along with their corresponding interactions (i.e., connecting edges). In that respect, the PDP extracts the required knowledge from the rules in order to provide the Planning Environment with the appropriate *Directives* that must be enforced on the workflow (making the Planning Environment an offline —in the sense that policy enforcement does not take place at real time— PEP), without implying that in the course of verification additional interaction with the PDP will not be needed, e.g., in case of detected inconsistencies.

A Bilateral Association corresponds to a pair of interacting tasks, including the interaction itself. The definition of a bilateral association is depicted in Figure 13. Bilateral associations are specified by means of the Compliance Metamodel; in fact, such an association can be considered as an elementary workflow consisting of two tasks and their interaction.

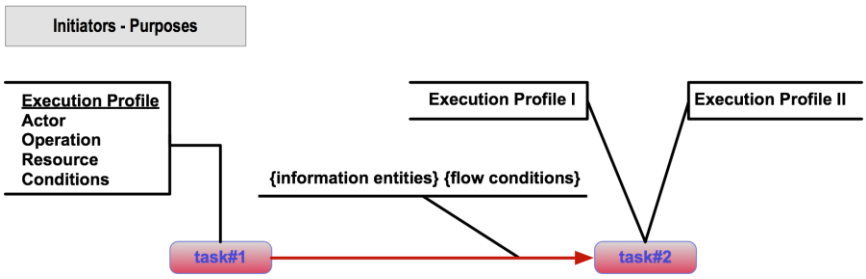

Figure 13: Bilateral Association

Compliance Metamodel Terminology: At a high level, the most fundamental artefacts of the Compliance Metamodel 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*[6]. Flows express dependencies between tasks, are represented through directed edges and are of two types: *control* and *data*. 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 both tasks continuously under execution, with t_B, however, being dependant on the stream of data produced by t_A. 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. Of special importance is 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. 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. Regarding modelling of flows, data and control edges share the same characteristics: each connects two tasks and denotes the flow direction, the information exchanged, the underlying conditions, and other flow properties; the distinction between them stems from the semantics of the connected operations regarding the manner they receive and consume information.

The *Bilateral Association Verification procedure* is mainly based on the output of the offline reasoning one described in Section 3.1 and it results in either discarding a given bilateral association or generating a set of authorisation directives that must be followed in order for the said association to incorporate all the security and privacy requirements prescribed by the Access and Usage Control Model. Specifically, there are five types of such directives:

Bilateral Validity Directives (BVDs): The generation of at least one such directive is a prerequisite for a bilateral association to be considered in principle valid. Each bilateral validity directive refers to a bilateral association as a whole, indicating a valid combination of actors—operation—resources for each of the involved tasks, as well as a valid association between them. The latter, in the simplest case, involves the definition of the edge joining the two tasks (transferred information entities, conditions, etc.), which may coincide or not with the one of the given bilateral association, while in some more complex cases it may additionally specify the insertion of one or more tasks, necessary for the compliant exchange of data between the two tasks (e.g., an encryption or other transformation task). In addition, such a directive refers to a valid purpose-initiator pair, associating its validity with the specific entities. However, validity directives may be complemented by directives of the other types prescribing additional requirements; thus, all other types of directives are always associated with some validity directive.

Task Presence Directives (TPDs): A task presence directive is used to express that a valid task as specified in a BVD requires the existence of another task or of a structure of tasks, possibly under certain contextual conditions, pre- and/or post- conditions. Apart from the required tasks, this directive additionally indicates their

[6] The term *asset* in workflows terminology corresponds to the access control term *resource*; for simplicity reasons the term *resource* is used throughout this document for describing both tasks and actions.

relative or absolute position within the association at the level of data exchange with respect to the reference task.

Task Forbiddance Directives (TFDs): Directives of this type prohibit the execution of some tasks in view of the presence of the ones specified within the BVDs. Thus, every such directive refers, on the one hand, to a BVD and, on the other, to the association's task that activates the prohibition, specifying the task/s with which the reference task conflicts, either in general or in case they are performed in some relative or absolute position with respect to the reference task. Similarly, a TFD may apply only under certain contextual conditions, pre- and/or post- conditions.

Input Requirement Directives (IRDs): An input requirement directive indicates that some task of the BVD needs to take some data as input. Thus, it points out the input data that the reference task should receive, along with the task or task structure that provides them, when necessary. The directive is complemented by the valid contextual conditions, pre- and/or post- conditions.

Input Forbiddance Directives (IFDs): As in the case of TPDs and TFDs, this directive type constitutes the negative analogue of IRDs, prohibiting the flow of specific information towards a task specified in a BVD.

In the context of this procedure, the PDP takes as input the considered bilateral association, along with the declared purposes it supposedly serves combined with the declared potential initiators, i.e. purpose-initiator pairs. The main steps of the verification procedure are the following:

1. Purpose compliance check
2. Extraction of the standalone valid tasks from the initial tasks contained in the considered bilateral association
3. Verification at the level of the bilateral association itself, as the latter has been adapted within the previous step

With the purpose-initiator pairs (*PIP*) being the starting point, at first the pairs consisting of leaf-elements of the corresponding *isA* graphs, i.e., purposes and roles that can't be further specialised, are extracted, which comprise the *LeafPurposeInitiatorPairs* set (*LPIP*). Next, the source (*src*) and destination (*dst*) tasks are extracted from the bilateral association (*ba*), as well as the operations associated with them (*BilateralAssociationOperations* set — *BAO*).

Thus, the afore-mentioned steps are performed for each *lpip* of the *LPIP* set, so that finally the resulting directives (*DIR* set) will refer to a specific purpose and a specific initiator. If such a pair is proved valid through the purpose compliance check, the next check concerns the standalone validity of each task initially included in the association; a task is considered standalone valid when it corresponds to at least one permitted action, meaning that the task is in principle valid for some specific purpose-initiator pair, without yet taking into consideration any potential constraints related to its execution or conflicts with third tasks. More than one standalone valid tasks may arise from each initial one comprising the *SAVST* and *SAVDT* sets, which contain the standalone valid tasks for the source and destination tasks, respectively. Members of these sets form one or more *bilateral meta-associations* (*BA'* set) based on the mapping of the standalone valid tasks to the initial ones.

Finally, for each of these meta-associations, containing now solely *in principle valid* tasks, the association per se is checked. This check involves several separate stages, like the verification of the interaction between the two

tasks and the identification of required tasks or of tasks conflicting with the ones of the bilateral association. During this check the authorisation directives for a given association are generated; no generated directives for an association means that it can't be executed whatsoever.

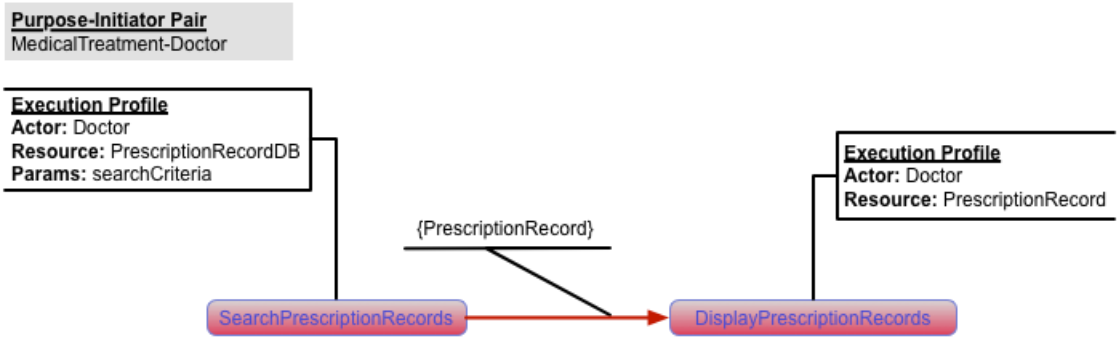

Figure 14: Bilateral association in the context of medical treatment provision

Figure 14 depicts a given bilateral association, which will be used in order to explain the afore-mentioned stages. The involved tasks concern the ePrescription scenario of Deliverable D2.1 "Use cases and requirements"; thus, in the context of the provision of medical treatment, an actor holding the role `Doctor` will first execute the operation `SearchPrescriptionRecords` with specific `searchCriteria` upon the related database of type `PrescriptionRecordDB` and then the results will be displayed to her through execution of operation `DisplayPrescriptionRecords` upon the received data of type `PrescriptionRecord`. The role `Doctor` constitutes the *initiator* of the association, while the latter serves the purpose of `MedicalTreatment`. Considering the two more specialised purposes `ePrescription` and `FirstAidProvision`, there are finally two leaf purpose-initiator pairs for which the validity of the given association will be examined (⟨`ePrescription, Doctor`⟩ and ⟨`FirstAidProvision, Doctor`⟩, meaning that the specified interaction may take place only for these purposes).

3.2.1.1 Purpose Compliance Check

The purpose compliance check verifies that the operations involved in the bilateral association may serve the declared purposes and that the roles of the initiator may act for these purposes, thus ensuring that the association is relevant and consistent with a purpose, while the purpose itself should not contradict with the access rights of the person initiating the association. In this context, two IMO properties are leveraged, namely `mayServePurpose` and `mayActForPurpose`, respectively; on the one hand, the bilateral association's operations (*BAO*) should be both associated to the pair's purpose through the `mayServePurpose` property, while, similarly, the pair's initiator role should be related to the same purpose by means of the `mayActForPurpose` property. In the case of the first leaf purpose-initiator pair of our example (⟨`ePrescription, Doctor`⟩), this would be valid if the operations `SearchPrescriptionRecords` and `DisplayPrescriptionRecords` may serve the `ePrescription` purpose and if the role `Doctor` may act for this purpose. It should be noted that the considered entities may be related to the purpose either directly, or indirectly due to IMO hierarchies.

3.2.1.2 Extraction of Standalone Valid Tasks

As afore mentioned, a task is considered standalone valid if it corresponds to at least one permitted action. A task may be proven valid as-is, or it may be transformed with some of its elements being replaced during this

stage with other relevant ones which are equivalent to the original ones and for which the access action is permitted. Additionally, any missing execution profile's elements are specified through this procedure. However, a task may be associated with more than one execution profiles, with each one reflecting one access action. Thus, in the general case, the mapping isn't one-to-one, while even a task with a single execution profile may lead to more than one permitted actions, as it will be described in what follows.

Based on the reference bilateral association of Figure 14, we will present the validity check and the subsequent extraction of the standalone valid tasks for the destination task. For this purpose, we search for permitted actions referring to the access action ⟨`Doctor`, `DisplayPrescriptionRecords`, `PrescriptionRecord`⟩, according to the execution profile's elements, valid for the `ePrescription` purpose. It is noted that, with a view to minimising the queries to the PDP, the final bilateral association should involve concepts as specialised as possible; in the case of the given access action, we assume that all its elements are completely specialised, thus making it already a leaf-action reflecting in turn specialised execution profiles. Figure 15 illustrates the steps for the extraction of the associated standalone valid task, starting from the originally given task of Figure 15.1. The first found permitted action of Figure 15.2 states that (under no special contextual conditions) the doctor may access a `PrescriptionRecord` only if the doctor has a care relationship with the owner of the resource[7], while the one of Figure 15.3 declares that in an emergency case a `PrescriptionRecord` will be displayed to the doctor regardless any constraints, e.g., upon the resource's ownership attribute. These two found `PermittedActions` instances result in the corresponding standalone valid tasks of Figure 15.4, with the constraint concerning the owner of the resource being added to the one of them due to the specified constraint of the corresponding enhanced entity (`Expr#100`).

In case a given task cannot be mapped to any permitted action, we search for permitted actions referring to less detailed resources or resources contained in the original one. Returning to our example, in case the permitted actions of Figure 15.2 and Figure 15.3 were not found, the next search would concern components) of the `PrescriptionRecord` (i.e., data types related through the `isPartOf` property to `PrescriptionRecord`), e.g., ⟨`Doctor`, `DisplayPrescriptionRecords`, `PersonalInformation`⟩, ⟨`Doctor`, `DisplayPrescriptionRecords`, `TheurapeuticProtocols`⟩ or ⟨`Doctor`, `DisplayPrescriptionRecords`, `Biometrics`⟩.

It becomes evident that multiple standalone valid tasks reflect differentiations in the execution of the operation of the originally given task, depending on constraints specified upon any access action's elements as well as contextual constraints, which may affect even the specification of the resource. Finally, it should be noted that partially specified tasks (e.g., tasks in the given bilateral association where no actor is specified) are accordingly populated through mapping to the found permitted actions' elements.

[7] Required pre- and post-actions will be separately examined in the context of task presence and task forbiddance directives extraction.

Figure 15: Validation of the $task_{dest}$ of bilateral association of Figure 14

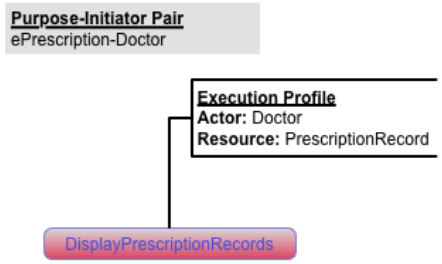

Figure 15.1: Original specification of the destination task

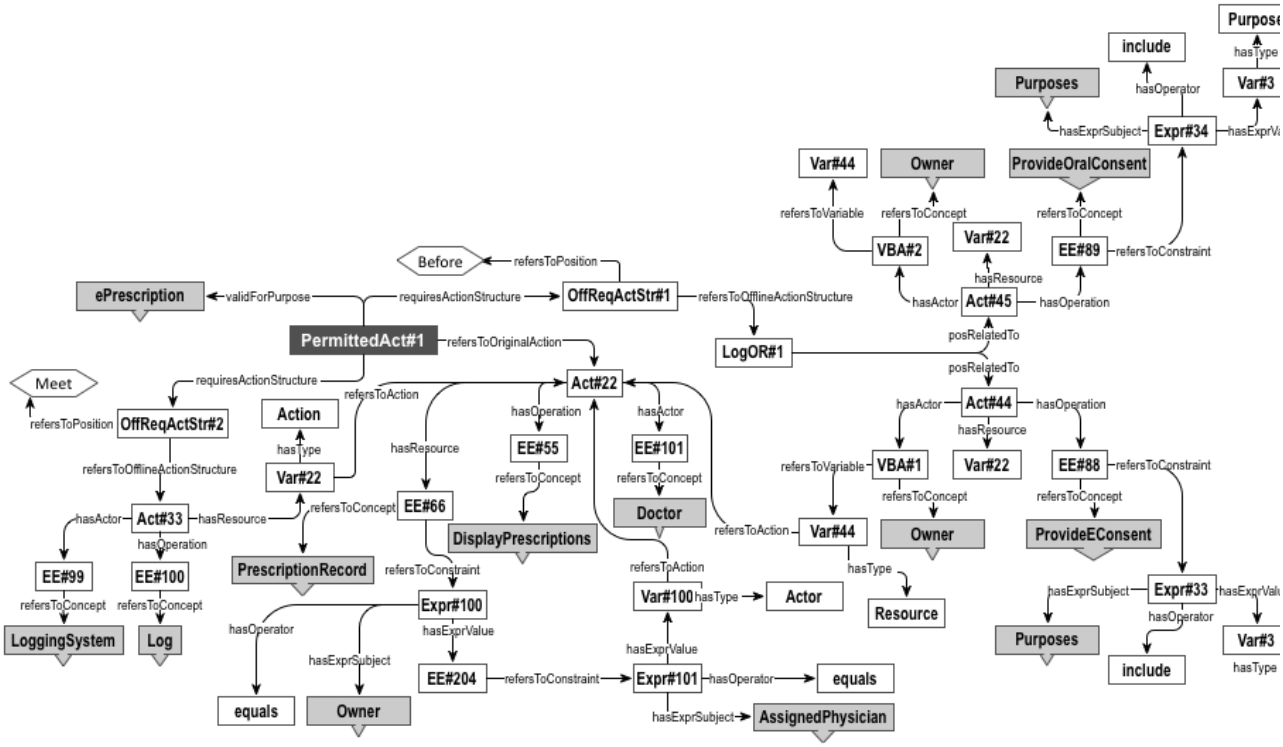

Figure 15.2: Found permitted action valid under any circumstances

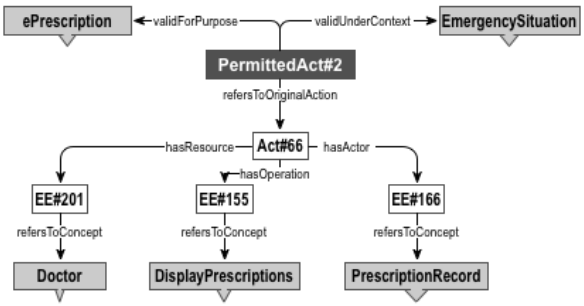

Figure 15.3: Found permitted action valid under an emergency situation; the same pre- and post-actions of Figure 15.2 apply here, however they are omitted for reasons of simplicity.

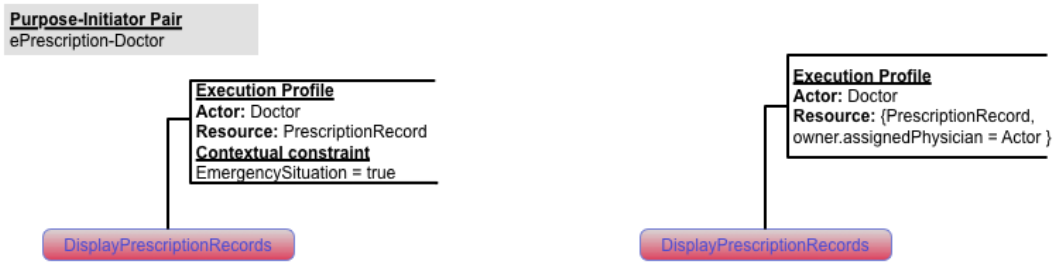

Figure 15.4: Standalone valid tasks for the found permitted actions

3.2.1.3 Verification at the Level of the Association

The last verification stage mainly concerns the validation of the bilateral association's edge, i.e., the validation of the interaction between the two tasks. In essence, the initial edge will be verified and accordingly transformed for each of the *meta-associations* as the latter have been formed at the end of the previous stage. For instance, the reference bilateral association leads to the formation of two meta-associations based on the standalone valid tasks for the given purpose-initiator pair ⟨`ePrescription, Doctor`⟩ (Figure 16), assuming that the source task was already standalone valid. Thus, in order for a bilateral association to be characterised as valid, at least one meta-association must be proven valid.

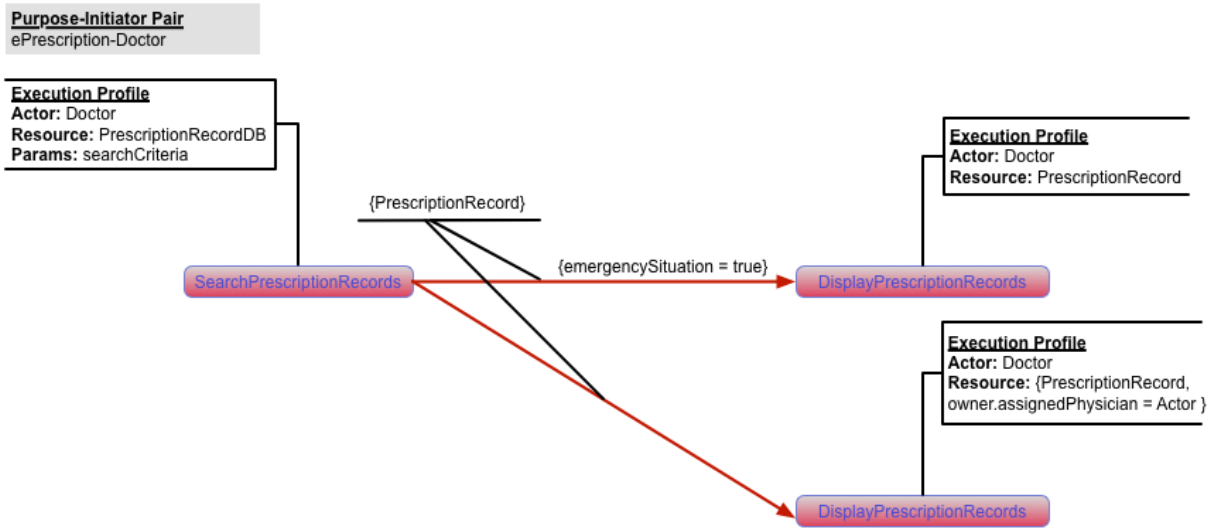

Figure 16: Bilateral meta-associations derived from the reference association

First, it should be ensured that there are no *definite conflicts* between the tasks of the bilateral (meta-) association, meaning that the execution of the one task doesn't preclude the execution of the other, regardless of any other factors like the presence or absence of third tasks. For the identification of such conflicts, the tasks are mapped again to permitted actions. In the general case, a definite conflict occurs when the reference action corresponds to the one task and there is an associated —by means of the `forbidsActionStructure` property— `OfflineRequiredActionStructures` instance corresponding to the other task with the appropriate position value. For instance, due to the permitted actions of Figure 15.2 and Figure 15.3 no conflicts exist between the tasks of any of the two meta-associations of Figure 16.

The validity of the control or data flow between the two tasks, i.e., the validity of the directed edge, constitutes another prerequisite for the validity of the considered association. In this context, it must be checked that each

one of the transferred data types is included in both the potential outputs of the source task and the potential inputs of the destination task, by checking that there is a path between the considered operation and the transferred data type (following the IMO `hasOutput` and `hasInput`, `refersToDataType` and `isA` properties, cf. Figure 3). In the simplest case, the transferred data type is included in both sets and the next check concerns the existence of read rights for the destination operation regarding the incoming transferred data type. Such read rights correspond once again to permitted actions while more than one such read rights may be found, as they may be valid under different contextual condition or for different required or forbidden pre-/post- actions, with each read right reflecting typically a different alternative edge for the considered bilateral association.

Regarding the meta-association of our example that will take place when there is no emergency situation, data have to be filtered before being displayed to the doctor, so that they concern only patients of this specific doctor. This means that an edge transformation is needed[8], that is, some other task(s) should be added between the association's tasks for transforming the output data of the source task to data that the destination task can or is permitted to read. Additions of this type are prescribed either directly by the specification of the flow through constraints on the attributes of the transmitted data and/or constraints regarding their *state*, or by the Semantic Access and Usage Control Model, due to restrictions imposed by the rules or for bridging differences between related data types. There are three main flow transformation patterns, which incorporate in the flow additional mechanisms for the security and privacy protection and lead to the addition of the following transformation tasks (that can be also combined):

- *Selection task.* This task implements the *selection* operation of relational algebra. Its addition is prescribed by constraints on the transferred data regarding the value of some attribute or contained data type, implying that only data meeting these filtering constraints will reach the operation op_{dst}; input and output data of this task are of the same semantic type and input data get filtered according to the desired criteria. The associated procedure is explained in Figure 17.
- *Projection task*. The *projection* operation of relational algebra implemented by this task is prescribed either by the read rights of op_{dst} or by the per se ability of op_{dst} to receive certain data. The output data of this task are of one or more semantic types contained (IMO `isPartOf` property) in the semantic type of the input data (see Figure 18).
- *State changing tasks*. Typical examples of this type of tasks are the encryption and anonymisation of data before reaching operation op_{dst}. The addition of such tasks is prescribed by *state changing constraints*, defined at the level of the bilateral association and/or the rules and specified using expressions which refer to the state of the transferred data. In this context, Figure 19.2 shows that the restrictions are imposed by rules through the associated permitted action. The constraint modelled through the expression `Expr#m` prescribes that data must be encrypted in a specific way, leading to the addition of the encryption task of Figure 19.3. It should be noted that such a transformation may also be applied only to specific parts of the transferred data.

[8] In case the associated with the task operation does not provide filtering capability.

Figure 17 Filtering the transferred data

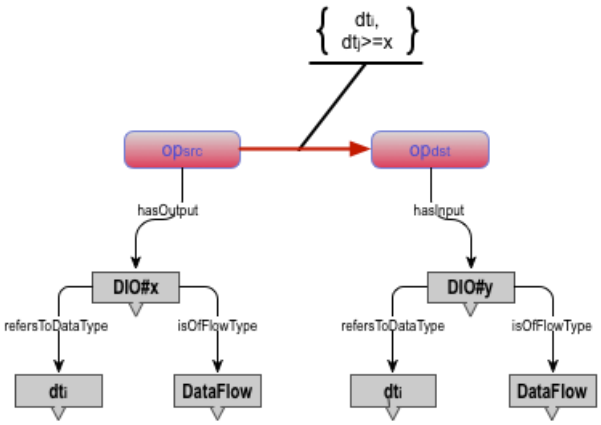

Figure 17.1: Input and output data compatibility check. A constraint on transferred data of type dt_i is defined already at the original bilateral association

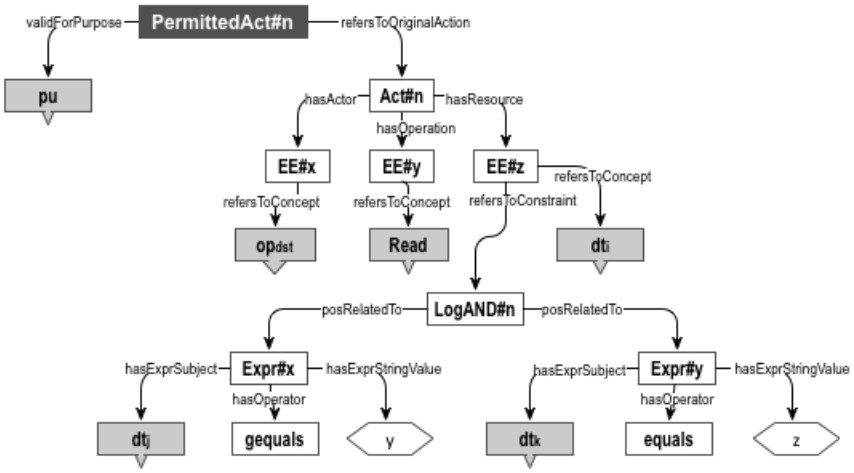

Figure 17.2: Read rights; dt_j and dt_k are related to dt_i through the *isPartOf* relation, and x>=y

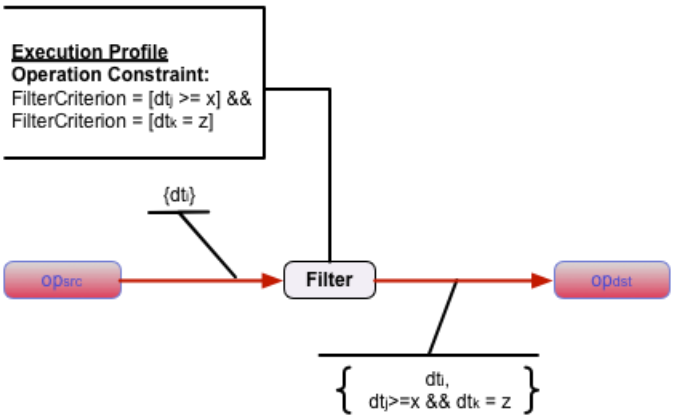

Figure 17.3: Selection task addition. Initial constraint on transferred data and constraint imposed by the `PermittedAct#n` have been merged

Figure 18: Projecting parts of data

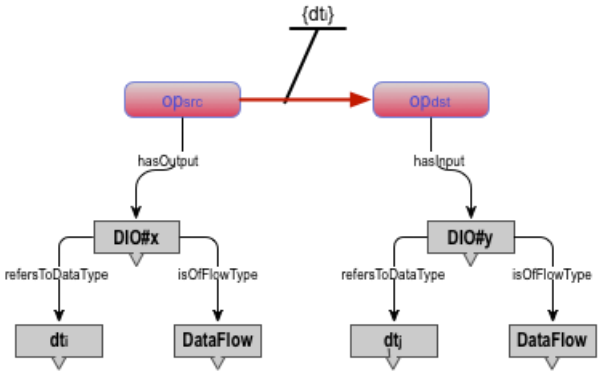

Figure 18.1: Input and output data compatibility check, where dt_i and dt_j are related through *isPartOf(dtj, dt_j)* relation

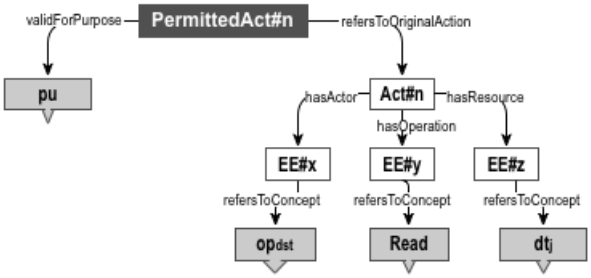

Figure 18.2: Read rights

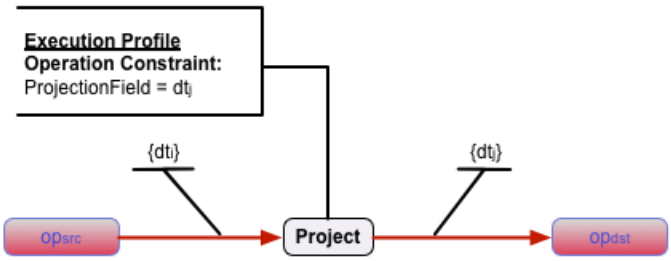

Figure 18.3: Projection task addition

Figure 19: Changing the state of transferred data

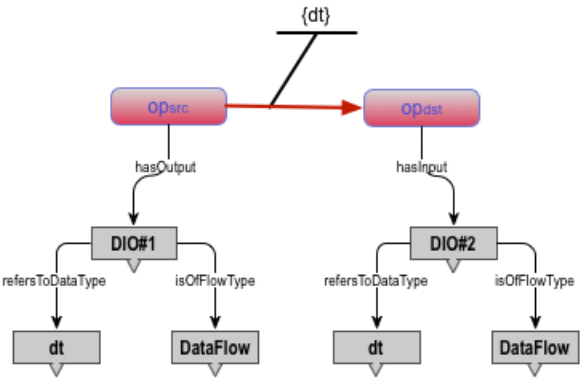

Figure 19.1: Input and output data compatibility check

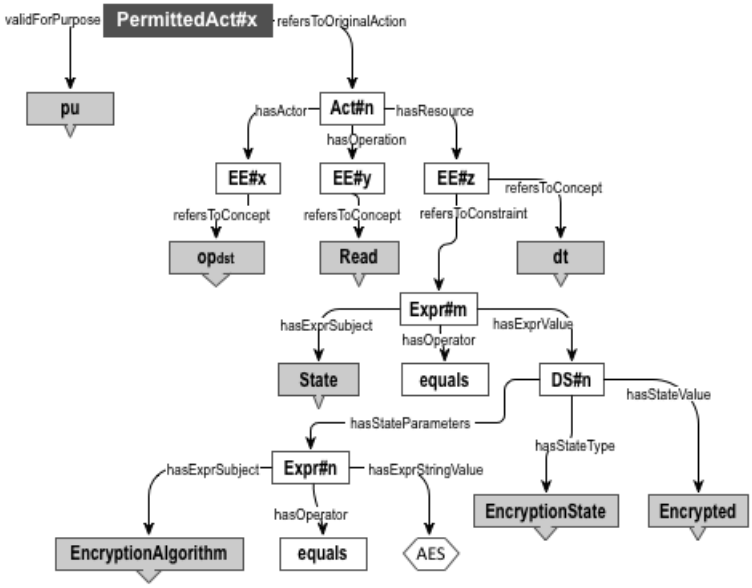

Figure 19.2: Read rights

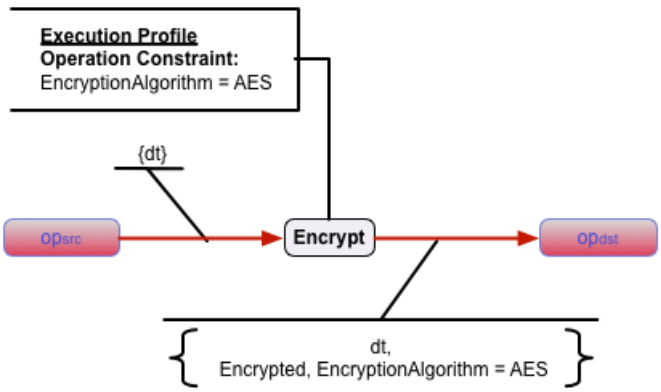

Figure 19.3: State changing task addition

With the completion of the afore described checks for all types of data associated with each alternative edge, a merging procedure is performed for defining the data types that will finally participate in the valid edge(s), which take into account both the contextual constraints imposed by the read rights and the flow constraints of the

initial edge. Thus, different or new branches of flow may occur, like in the reference example where different standalone valid tasks exist for different contextual conditions. At this point, each valid specification of a bilateral association, which may now include more than two tasks, is translated into a BVD describing in detail the valid implementation of the initially given tasks and edge. This directive is possibly complemented by directives of the other types, prescribing the presence or absence of third tasks, as well as additional or forbidden flow towards the tasks of the association. Therefore, in this last stage of the bilateral association's verification, all the tasks and edges of the valid association, as it is described in the corresponding BVD, are mapped again to permitted actions, taking into consideration the current purpose-initiator pair and contextual conditions, and the associated `OfflineRequiredActionStructures` instances are in turn translated to the corresponding tasks and flows specification. For instance, for the valid meta-associations of Figure 16, mapping of the destination tasks to the `PermittedAct` instances of Figure 15 implies task structures that must be executed before and immediately after the reference task, respectively, because of the required action structures `OffReqActStr#1` and `OffReqActStr#2` and lead to the generation of the corresponding TPDs. Figure 20 illustrates the transformation of the given bilateral association of Figure 14, that will be communicated to the planning environment by means of the afore-mentioned directives, with a view to adapting processes and operations in order to become compliant, in the context of the verification procedure.

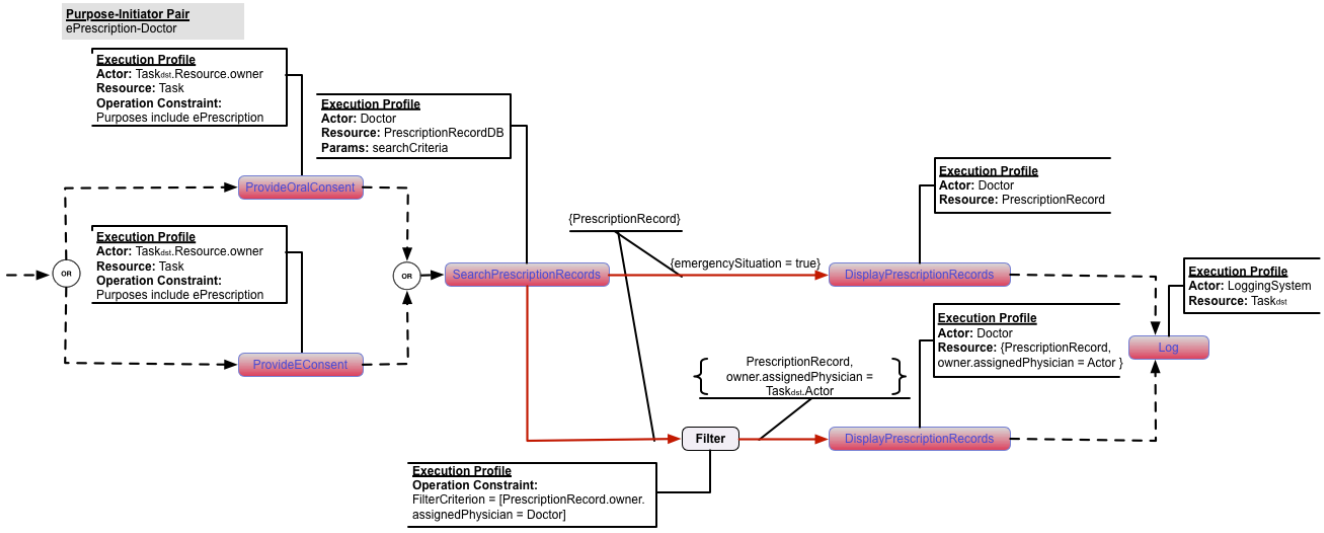

Figure 20: Transformation of the bilateral association of Figure 14

3.2.2 Evaluation of real-time access requests

As afore-mentioned, real-time requests are submitted to the PDP by the Data Management middleware (or by any other component exposing PEP functionality) through the Authorisation interface (cf. Figure 21). Such requests contain information about all the involved entities of the attempted access action; for instance, in case a doctor requests access to a prescription record, the PDP will evaluate the request based on the purpose of access and the attributes that the doctor, the processing operation and the prescription record bear, e.g., the `UserId` of the doctor in the context of the ePrescription system, the `Owner` of the prescription record, as well as attributes that may not be available at the time of request. The latter concern mostly nested attributes of involved entities, like the attribute `AssignedPhysician` associated with the owner of the prescription record.

It becomes evident that, in order for the PDP to evaluate requests of this kind, further interactions with other components are needed. On the one hand, the PDP may need to ask the Data Management middleware about necessary information that is not contained in the original request, as in the case of nested attributes of our example. Additionally, in order to check that a pre-action has or has not been executed, interaction with a *Reference Monitor* is deemed necessary. These two functionalities will be offered by a *Context Handler*, in line with the XACML paradigm, providing information external to the PDP and PEP.

Evaluation of real-time requests is once again based on the concept of offline-extracted permitted actions. Returning to the permitted action of Figure 15.2, the PDP will first check if constraints upon the resource `PrescriptionRecord` are satisfied; if not, further checks are performed. Similarly to the process for extracting standalone valid tasks described in Section 3.2.1.2, access rights to partial or less detailed information will be examined. If access to parts of the resource of less detailed resources is also denied, a negative response is returned to the Data Management middleware.

Satisfaction of constraints will result in the subsequent check for pre-actions that should or should not have preceded; in that respect, the PDP has to be informed about prior execution of either of the actions $\langle X,$ `ProvideEConsent`, $\langle$`Doctor`, $\texttt{DisplayPrescriptionRecords}_{\texttt{Purposes include ePrescription}}$, $\texttt{PrescriptionRecord}_{\texttt{Owner} = X}\rangle\rangle$ and $\langle X,$ `ProvideOralConsent`, $\langle$`Doctor`, $\texttt{DisplayPrescriptionRecords}_{\texttt{Purposes include ePrescription}}$, $\texttt{PrescriptionRecord}_{\texttt{Owner} = X}\rangle\rangle$, with variable X representing the owner of the prescription record that the doctor requested to access for the purpose of `ePrescription`. A positive answer by the Context handler will in turn result to an authorisation response returned by the PDP to the Data Management middleware along with a structure describing the obligatory post-condition of logging the access action $\langle$`LoggingSystem`, `Log`, $\langle$`Doctor`, $\texttt{DisplayPrescriptionRecords}_{\texttt{Purposes include ePrescription}}$, $\texttt{PrescriptionRecord}_{\texttt{Owner} = X}\rangle$, so that the Data management middleware will instruct the related system to perform the required post-action.

In case it turns out that the required pre-actions have not taken place or forbidden pre-actions have taken place during the system operation, access will finally be denied. However, in cases like the one of the reference example where the pre-action (provision of consent) is interrelated with the access action itself (the access action constitutes the resource of the pre-action), further checks are required; the PDP will request the Context Handler if the data subject X has provided partial consent, e.g., if one of the following actions has taken place $\langle X,$ `ProvideEConsent`, $\langle$`Doctor`, $\texttt{DisplayPrescriptionRecords}_{\texttt{Purposes include ePrescription}}$, $\texttt{PersonalInformation}_{\texttt{Owner} = X}\rangle\rangle$, $\langle X,$ `ProvideEConsent`, $\langle$`Doctor`, $\texttt{DisplayPrescriptionRecords}_{\texttt{Purposes include ePrescription}}$, $\texttt{TheurapeuticProtocols}_{\texttt{Owner} = X}\rangle\rangle$ or $\langle X,$ `ProvideEConsent`, $\langle$`Doctor`, $\texttt{DisplayPrescriptionRecords}_{\texttt{Purposes include ePrescription}}$, $\texttt{Biometrics}_{\texttt{Owner} = X}\rangle\rangle$[9] in the case of `ProvideEConsent` operation. If partial consent has been provided, the PDP will search for permitted actions concerning access to the related part of the `PrescriptionRecord` and accordingly respond to the Data Management middleware, which in turn will have to instruct based on the PDP's response the appropriate components in order to e.g., filter/anonymise/encrypt information of the `PrescriptionRecord` that should not become available to the requestor .

[9] `PrescriptionRecord` has been replaced with data types related through the `isPartOf` relation

Access to a resource different than the requested one may emerge even in the case of a verified and adapted by the Planning Environment process, as variables in the access and usage control rules may only be evaluated in runtime. Such deviations from already compliant (at specification level) processes will be examined by the Process Monitoring component; checking a posteriori the compliance of running processes will help to identify discrepancies between modelled and observed behaviour, but also follow and guide process evolution over time, for the benefit not only of legality but also of the affected businesses per se.

4 Governance architecture

The rule-based framework presented in the previous sections is the core BPR4GDPR component as regards governance of the whole system. The governance module of the system is being designed with a dual focus: on the one hand, to be able to extract precise information from complex —sometimes even conflicting—rules, and, on the other hand, to be efficient for coping with stringent performance needs of real-time operations.

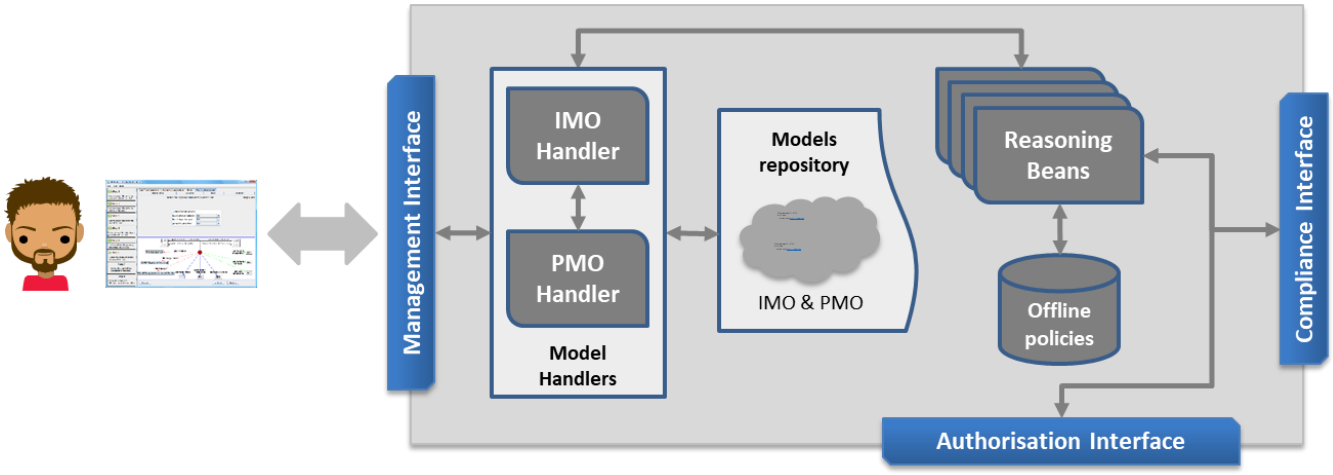

Figure 21: Governance component logical architecture

In Figure 21, the logical architecture of the governance component is sketched. As illustrated, it internally consists of the following modules:

Models Repository. This module provides hosting to the two fundamental ontological models, i.e., the Information Model Ontology and the Policy Model Ontology. In this context, 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 accessible by the other components, particularly the Reasoning Beans and the Management Interface (see below). In this context, it offers all functions necessary for the management of the models, such as graphs' configuration and rules' administration.

Reasoning Beans. Reasoning constitutes the most essential part of a rule-based system, as it implements the knowledge extraction and decision-making functions. In BPR4GDPR, the underlying intelligence is offered by the Reasoning Beans, that implement the algorithms for answering all queries regarding authorisation and compliance. They constitute the PDP of the system, comprising a dynamic set of stateless engines —for improved performance— that extract knowledge from the ontological models and feed the other components with instructions, addressing either authorisation decisions or compliance patterns and guidelines.

Offline Policies. Motivated by the need for improved performance during real-time operation, BPR4GDPR leverages the paradigm of offline reasoning, i.e., proactive extraction of knowledge contained in the access and usage control rules. Through the Offline Reasoning procedure described in Section 3.1, all the required knowledge becomes available already by the request time, thus offering performance gains. In other words, all the heavy processing tasks are performed offline and only when the PMO is updated, for instance, when new rules are added or existing ones are revoked.

In order to interact with the other BPR4GDPR components, the governance module provides three interfaces, devised for, respectively, answering authorisation queries, responding to compliance-related queries, and management aspects of the ontological models.

Authorisation interface. This interface serves the fundamental need of providing knowledge related to authorisation to other components of the BPR4GDPR ecosystem, particularly the Data Management Bus that holds the role of being the primary PEP (cf. Section 3.2.2).

Compliance interface. This interface is devised to provide other components, particularly the planning environment (to be documented in the Deliverable D4.1 "Initial specification and prototyping of the process re-engineering framework"), with instructions as regards the adaptation of processes and operations in order to become compliant, as part of the verification procedure (cf. Section 3.2.1).

Management interface. Through this interface, the appropriate functions are provided for the management of the Information Model and the Policy Model. This includes the configuration of the graphs and the specification of the rules (cf. Section 5).

5 User interfaces

Whereas what has been presented in the previous sections comprises the core of the policy framework —how policies are formalised and how decisions are made thereof—, the interaction with the user constitutes a challenging issue. Indeed, visualisation of access and usage control aspects and underlying resource models, and enabling policy administrators to work in an intuitive manner, is emerging as an important research topic ([31], [32], [33], [34], [35], [36]).

BPR4GDPR is a project that is putting special emphasis on SMEs with limited resources. In such environments, the consideration of the human factor, in terms of intuitive interaction with the systems, is critical for achieving the maximum impact of the underlying technology. To this end, the policy framework is complemented by a user interface that aims at facilitating information model and policy administration.

5.1 Information Model administration

For the administration of the Information Model, a user-friendly software application has been developed, devised for the specification and management of the underlying semantic model. This tool, the *editor* (Figure 22), provides all necessary functionality for the management of all different information classes and their relations, and the corresponding ontology, as described in Section 2.1. The use of the editor hides the technical details of the model, requiring no particular technical expertise by its users; it translates the input provided through the graphical interface to machine code.

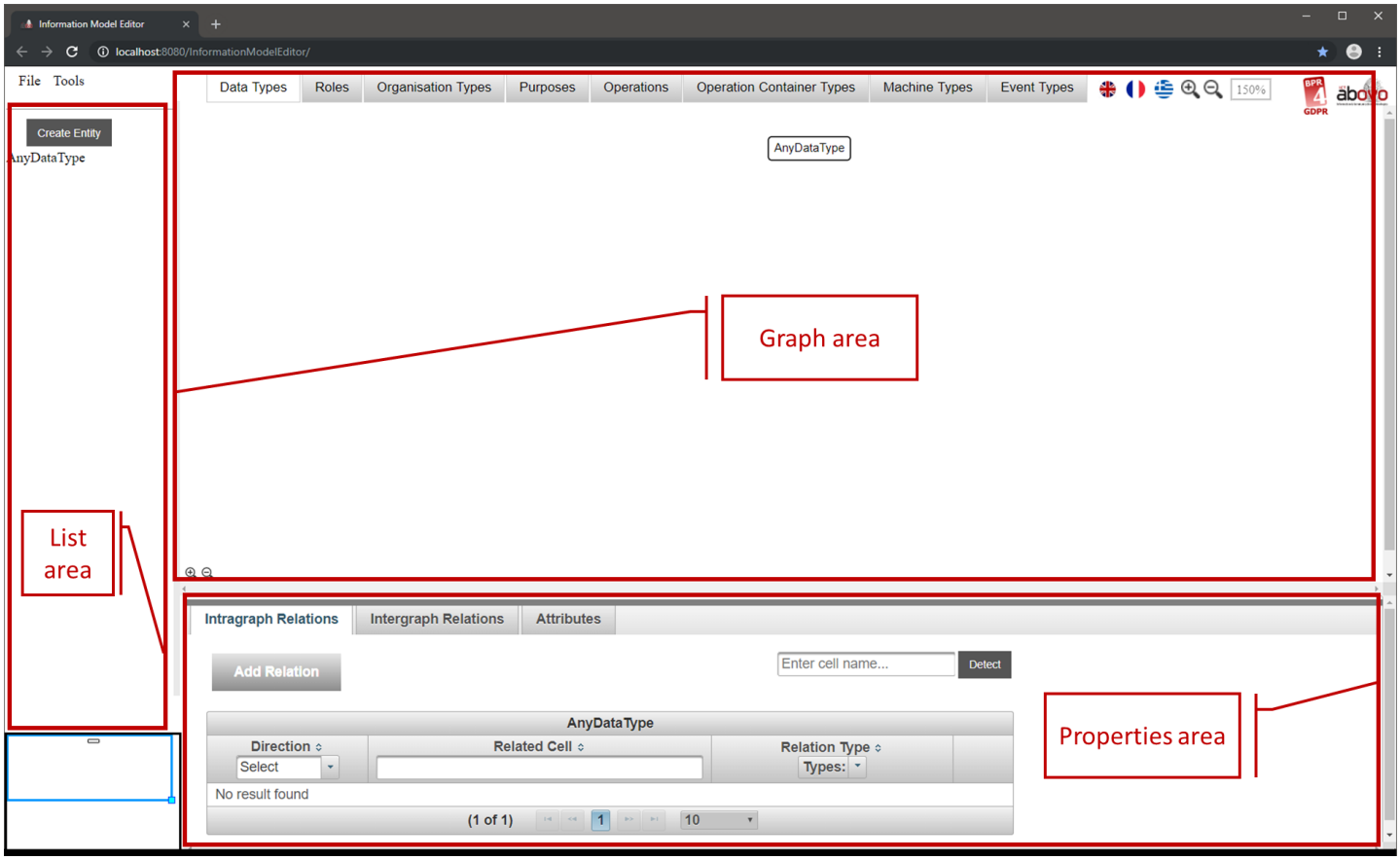

Figure 22: Information Model Editor

As illustrated in Figure 22, there are three main areas the user sees upon logging in, notably the *Graph*, *List* and *Properties* areas. The Graph area allows graphically editing the different information classes, including the definition of entities and the hierarchies thereof. As shown in the upper part of this area, it is organised in *tabs*, each used for the illustration of a different graph of the Information Model. The List area provides a list of the entities, sorted alphabetically, in order to facilitate navigation. The Properties area is devised for the management of entities' properties, notably attributes and relations, both inside the same graph (i.e., hierarchy) and relations spanning across different graphs (e.g., the association of an operation with sought purposes). The user interface is complemented by some control functions, as well as a *map* for making easier the navigation in large graphs.

As an example, Figure 23 illustrates (part of) the Roles' graph of a model. As shown, there are multiple entities in the graph, interconnected with two types of relations; blue lines denote `isA` relation, whereas green lines reflect the `isPartOf` relation. Since the entity selected by the user is the `BoardOfDirectors`, the Properties area lists the ones characterising this entity. To this end, there are shown various incoming `isPartOf` relations (with `ChiefExecutiveOfficer`, `BoardMember`, etc.), as well as one outgoing `isA` relation, with `Executive`; through the latter, the `BoardOfDirectors` entity is indirectly connected with the `AnyRole`.

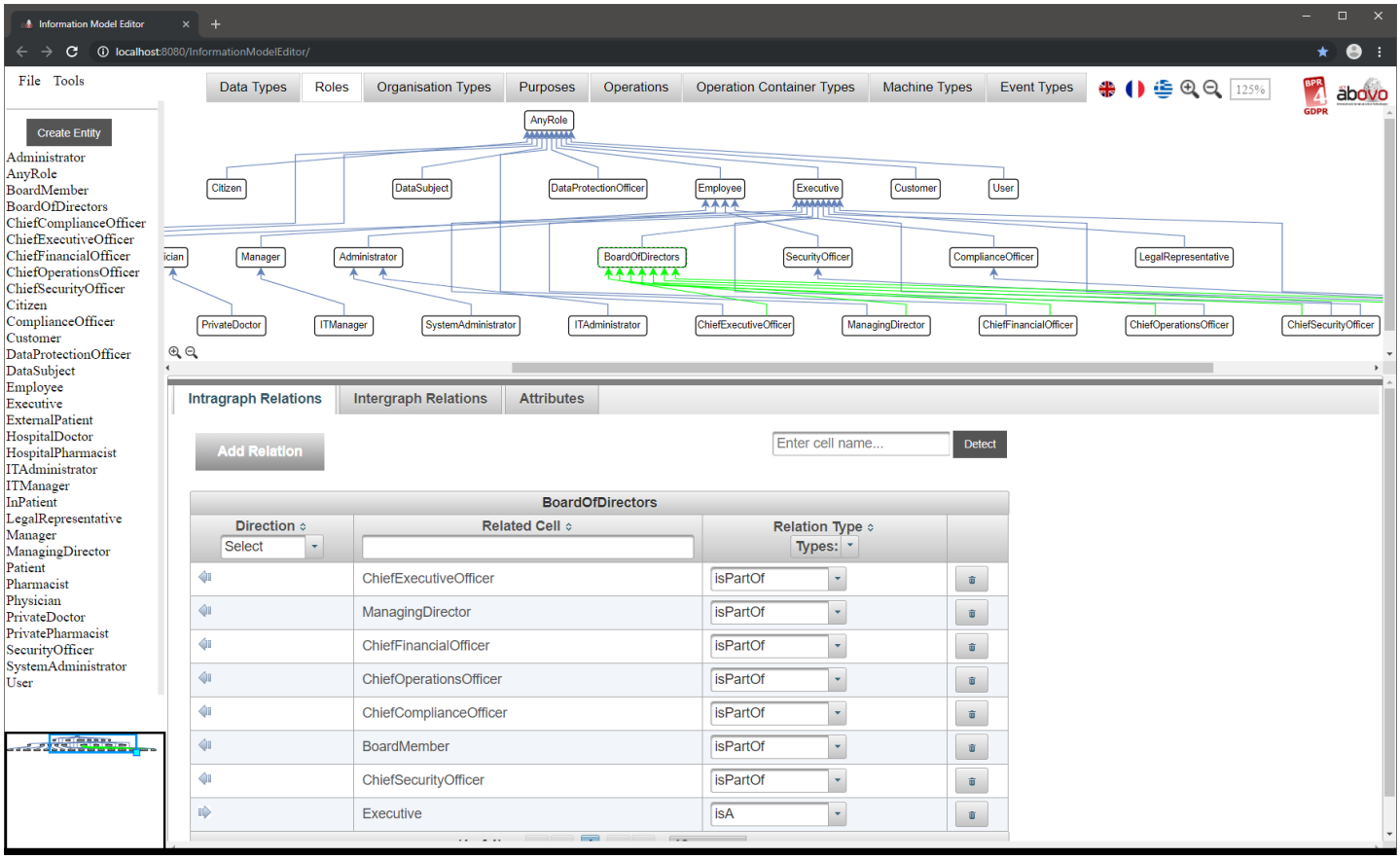

Figure 23: Roles graph example

As regards its functional features, there have been two priorities: user-friendliness and flexibility. As regards the former, a fact to note is that most administration operations can take place in multiple ways, typically graphically and using appropriate forms. For instance, in order to create a relation between two entities, one way is by

drawing a line connecting these entities, whereas the second way, illustrated in Figure 24, is implemented by pressing the “Add Relation” button and, thereupon, selecting the appropriate values in the form; Figure 24 also shows the use of auto completion, leveraged by the editor whenever applicable.

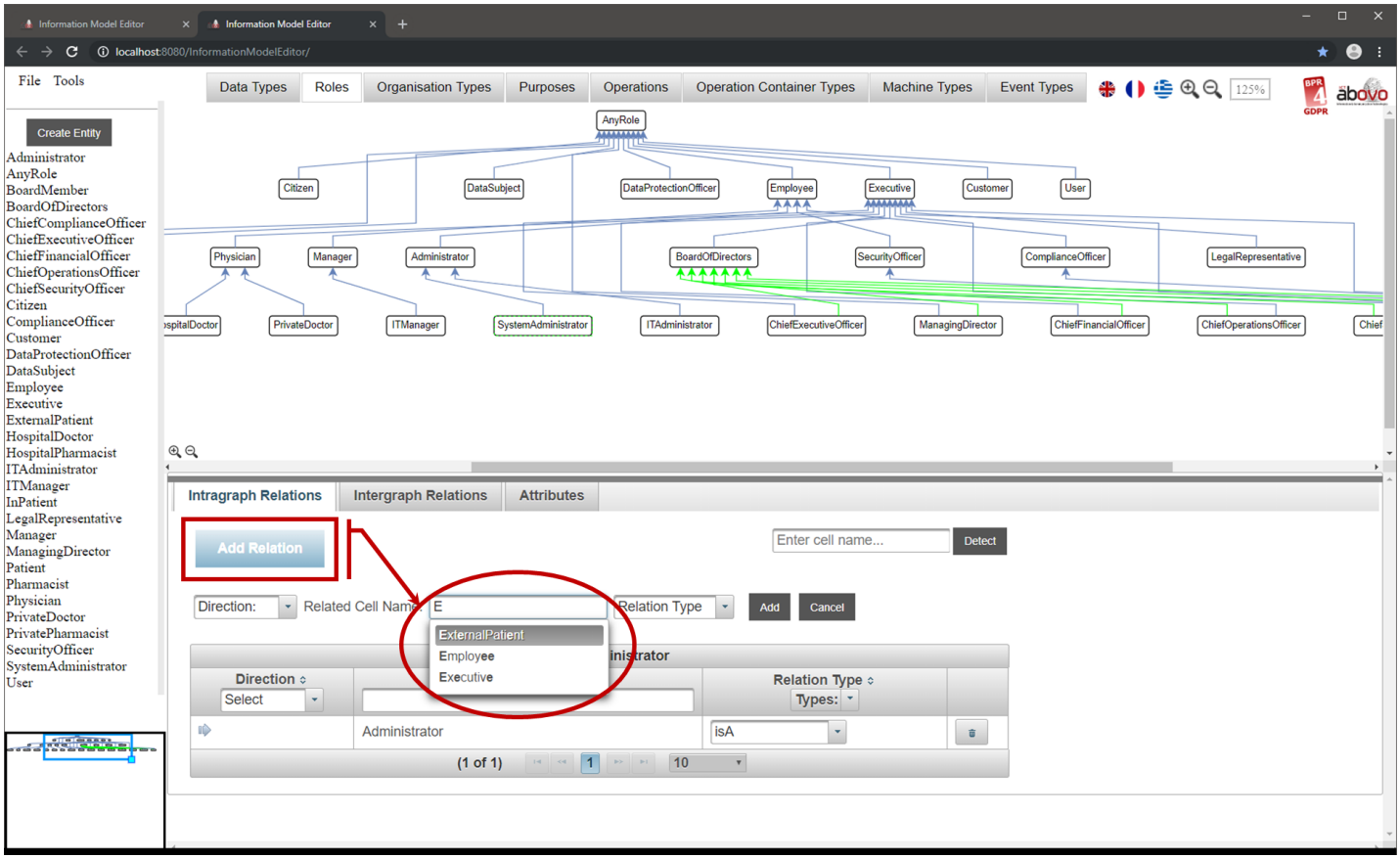

Figure 24: Example of new relation insertion using a form

As regards flexibility, it is to be noted all parameters of the editor are configurable. This does not only concern the look-and-feel of the interface and contextual parameters (e.g., the language), but even the classes of the information model and the relations thereof. To this end, also the underlying API is model agnostic, in the sense that all functions therein are neutral and take the name of the classes as a parameter in each call. This way, the editor can be adapted to the information model of any organisation and allows supporting any special need and particularity, whereas the API is enabled to correctly behave at any corresponding call.

As an example, Figure 25 presents an excerpt of a sample configuration file[10]. In this listing, line #2 denotes the existence of a “Roles” class and, consequently, a tab in the editor; the name of the tab (“Roles”) is the parameter to be used in all interactions with the underlying API. Line #3 states that the label to be used for the tab is “Roles”, whereas line #4 indicates the default root of the graph, from which all entities inherit; to this end, in Figure 22, although the graph is initially empty, one can see under the “Data Types” tab the `AnyDataType` entity already and by configuration present. Lines #5 to #14 simply set some look-and-feel parameters. Relations are defined in lines #15 to #21. Specifically, intra-graph relations are declared in lines #16 (`isA`) and #17

[10] Currently, configuration is based on JSON files; however, the plan is to migrate to a database, in order to enable per organisation differentiation, as well as modification of the configuration without the need to deploy the application again.

(`isPartOf`), along with the colour to be used for their representation; note that no further details are provided, since these two relations are also defined at the back-end API level as the fundamental particularisation and inclusion properties. On the other hand, line #20 defines that a role may be related with one or more purposes, via the `mayActForPurpose` (cross-graph) relation; therefore, when "right-clicking" at a role entity in the graph area (Figure 26), the corresponding option is provided in the context menu, denoting also the range of values ("Purposes").

```
1   {
2       tabname: "Roles",
3       tab_label: "Roles",
4       super_instance: "AnyRole",
5       shape: "rectangle",
6       rounded: "true",
7       spacing: "2",
8       strokeColor: "black",
9       gradientColor: "white",
10      fontColor: "black",
11      verticalAlign: "middle",
12      fontSize: "10",
13      edgeStyle: "elbowEdgeStyle",
14      jettySize: "jettySize",
15      relations: {
16              isA: "#6482B9",
17              isPartOf: "#00FF00"
18      },
19      crossTabRelations: {
20              Purposes: "mayActForPurpose:Purposes"
21      }
22  }
```

Figure 25: Roles tab configuration

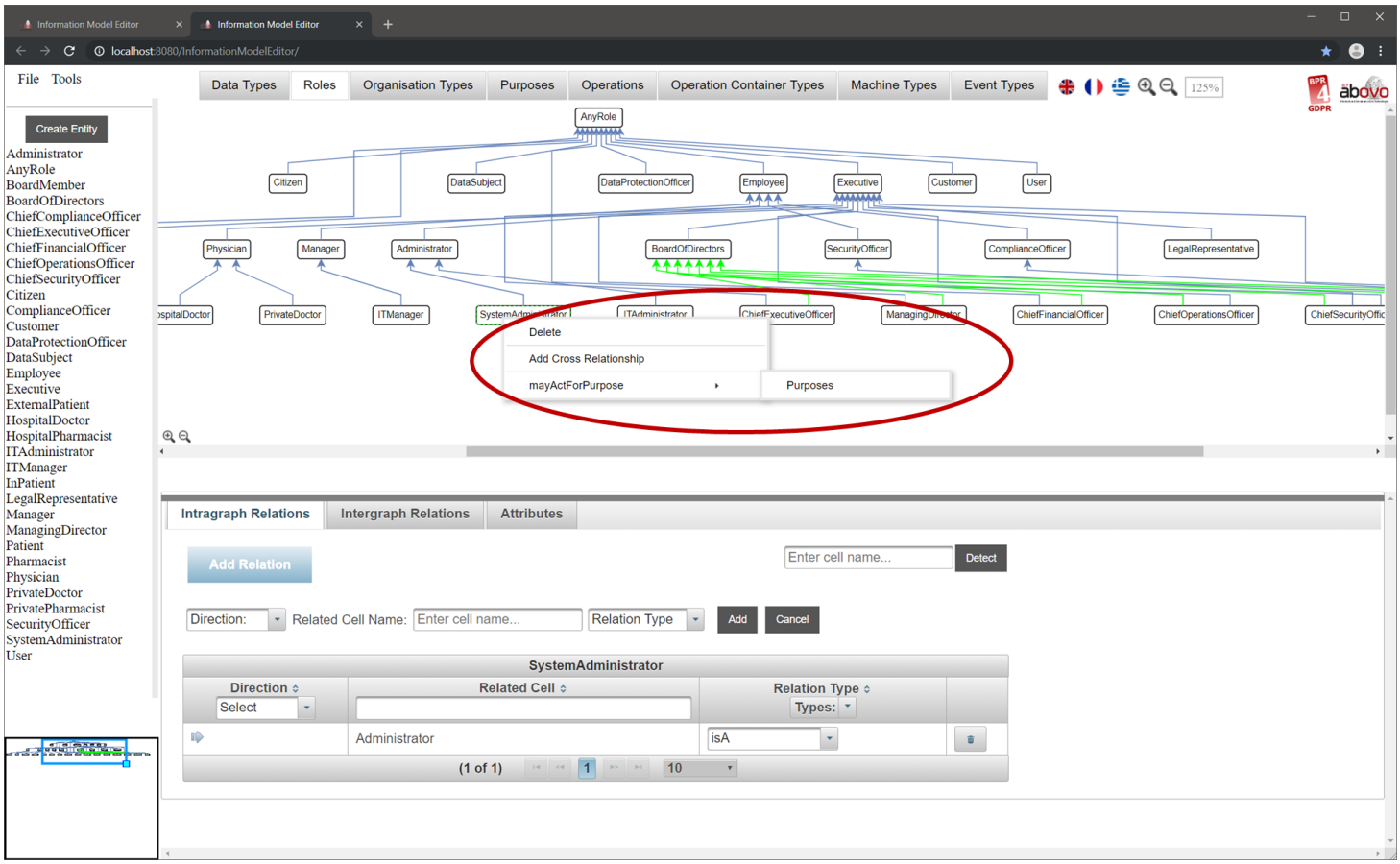

Figure 26: Cross-graph relation context menu

5.2 Policy Model administration

The user interface for the policy model administration is currently under development. In fact, authoring and management of such expressive but complex rules may be a very challenging task, implying the need for a trade-off between expressiveness and user friendliness.

The respective editor is going to be integrated with the one for Information Model administration, as rules are defined upon its entities. The editor will obviously provide functionality for creating, editing, deleting and listing rules, as well as rules' searching by means of advanced search criteria, while it will guide the user through the definition of a rule's all elements in a user-friendly manner; for instance, upon the definition of an action's resource, auto completion of resource's associated attributes will facilitate the definition of related constraints. Generation of meta-rules upon the creation of a high-level rule and the subsequent conflict resolution will take place transparently to the user, who will be only informed about the successful insertion of the rule (respectively for the cases of rule editing and deletion).

User-friendliness will be further fostered through the introduction of different views implying different grouping of rules; a *resource-centric view* will inform the user about access rights on specific resources, answering queries like "*who may access a resource of a specific type*", while an *actor-centric view* will display access rights grouped by the access requesting entity, e.g., querying "*what are the resources that can be accessed by a Doctor*". It is anticipated that also navigation between the Policy Model and Information Model administration editors will be supported; for instance, selecting a specific data type in the Information Model editor will display the access and

usage control rules associated with it (with the specific data type being the resource in an access, pre- or post-action), as a fourth tab in the Properties area (cf. Figure 22), while clicking on a data type involved in a rule while in the Policy Model editor will navigate to the Data Types tab of the Information Model editor with the Properties area listing properties that characterise the specific data type.

It is noted that a first fully-functional version of the rules management API has been implemented, supporting creation and deletion of rules, along with the generation of meta-rules. In the current implementation, the "Deny Override" algorithm [37] is supported in the context of rule combination and conflict resolution; in the final version, it is anticipated that all XACML rule combination algorithms will be supported, with the administrator of the system being able to select the preferred one.

6 Conclusions

This Deliverable documented the BPR4GDPR policy framework, aiming at regulating the BPR4GDPR operations by means of security and privacy policies that reflect the GDPR and related legislation. To this end, the semantic access and usage control model was presented in detail, along with the necessary mechanisms for knowledge extraction and decision making that will drive the compliant execution of operations; specifically, the BPR4GDPR policy framework provides the means for system governance in real-time, in the sense that it sets the rules that regulate the operation of BPR4GDPR components, while, on the other hand, policies comprise the knowledge base that feeds the procedure of process re-engineering, towards compliant by design process models.

In the next phase of the WP3 and towards final prototyping of the policy framework, special focus will be given to the fine-tuning of knowledge extraction algorithms according to any further requirements that will arise during integration with the PEPs (through the Compliance and Authorisation interfaces), implementation of the Context Handler module, finalisation of the Management interface and implementation of the Policy Model administration user interface.

References

[1] Information technology -- Role-Based Access Control (RBAC), INCITS 359--2004, ANSI/NIST, 2004.
[2] E. Papagiannakopoulou et al., “Privacy-Aware Access Control”. In Enc. of Infor. Science and Technology. IGI Global. 2014.
[3] Byun, J.-W., & Li, N., Purpose based access control for privacy protection in relational database systems, The VLDB Journal 2008.
[4] Masoumzadeh, A., Joshi, J. B. D., PuRBAC: Purpose-aware role-based access control, in Proceedings of OTM 2008.
[5] Ni, Q. et al., Privacy Aware Role Based Access Control, In Proceedings of SACMAT 07.
[6] Casassa Mont, M., Dealing with Privacy Obligations: Important Aspects and Technical Approaches, TrustBus 2004, 2004.
[7] Hilty, M. et al., On obligations. In Proceedings of ESORICS 2005.
[8] Cuppens, F., Cuppens-Boulahia, N., Modeling Contextual Security Policies, IJISS 7(4), 2008.
[9] Joshi, J. B. D. et al., “A generalized temporal role-based access control model”, IEEE TKDE, 17(1), 4–23, 2005.
[10]Bertino, E. et al., GEO-RBAC: A spatially aware RBAC, In Proceedings of the SACMAT ’06, 2005.
[11]Ravari, A. N. et al., GTHBAC: A generalized temporal history based access control model, Telecom. Systems, 45(2) 2010.
[12]J.D. Ultra, S. Pancho-Festin, A simple model of separation of duty for access control models, Computers & Security, Vol.68, 2017.
[13]Joshi, J. B. D. et al., Dependencies and Separation of Duty Constraints in GTRBAC, In SACMAT '03.
[14]Hu, V. et al., Guide to attribute based access control (ABAC) definition and considerations, Special Pub. 800–162 U.S. DoC, 2014.
[15]Yuan, E., Tong, J., “Attributed based access control (ABAC) for web services”, in Proceedings of the ICWS '05, 2005.
[16]A. Cavoukian et al., The Importance of ABAC: Attribute-based Access Control to Big Data: Privacy and Context, 2015.
[17]Zhang, X. et al., Formal model and policy specification of usage control, ACM Trans. Inf. Syst. Secur. 8 (4) (2005) 351–387.
[18]Park, J., Sandhu, R., Towards usage control models: beyond traditional access control, in SACMAT ’02.
[19]Zhang, G., Gong, W., “The research of access control based on UCON in the internet of things”, J. Softw. 2011.
[20]OASIS, “eXtensible Access Control Markup Language (XACML) Version 3.0.”, OASIS Standard, August 2010.
[21]ANSI/NIST, Information technology - Next Generation Access Control - Functional Architecture (NGAC-FA), INCITS 2013.
[22]D. Ferraiolo et al., NISTIR-7987 Revision 1, “Policy Machine: Features, Architecture, and Specification”, October 2015.
[23]D. F. Ferraiolo et al., NIST SP-800-178, A Comparison of Attribute Based Access Control Standards for Data Services”, 2015.
[24]Ayed, S. et al., Deploying security policy in intra and inter workflow management systems. In ARES 2009.
[25]Russello G. et al., A workflow-based access control framework for e-health applications, in: WAINA 2008.
[26]Alam M. et al., Constraint based role based access control in the sectet-framework a model-driven approach, J Comput Secur 2008.
[27]Menzel M, Meinel C, SecureSOA, In: IEEE international conference on services computing, 2010.

[28]E. Papagiannakopoulou et al., Leveraging Ontologies upon a Holistic Privacy-aware Access Control Model, FPS'2013.

[29]Knuplesch, D. et al., A visual language for modeling multiple perspectives of business process compliance rules, SoSyM 2017

[30]M. N. Koukovini et al., Towards Inherent Privacy Awareness in Workflows, in DPM 2014.

[31]C. Morisset, D. Sanchez, “On Building a Visualisation Tool for Access Control Policies”, in ICISSP 2018.

[32]I. K. Tanoli, et al, “Towards automatic translation of social network policies into controlled natural language”, in Proceedings of IEEE RCIS 2018.

[33]I. Matteucci, et al, “A Design Phase for Data Sharing Agreements”, in Proceedings of DPM 2011.

[34]A. Mousas, et al, “Visualising Access Control: The PRISM Approach”, in PCI 2010.

[35]M. Anwar, P. Fong, “A visualization tool for evaluating access control policies in facebook-style social network systems”, in ACM SAC 2012.

[36]P. Rao, et al, “Visualization for access control policy analysis results using multi-level grids”, in IEEE POLICY 2009.

[37]P. Mazzoleni, et al, “XACML Policy Integration Algorithms”, ACM Trans. Inf. Syst. Secur. 11, 1, Article 4, 2008.