A Workflow Reference Monitor for Enforcing Purpose-Based Policies

Mohammad Jafari
jafarm@ucalgary.ca
Jörg Denzinger
denzinge@cpsc.ucalgary.ca
Reihaneh Safavi-Naini
rei@ucalgary.ca
Ken Barker
kbarker@ucalgary.ca
Department of Computer Science, University of Calgary

Technical Report: 2013-1046-13

September 2013

Abstract

Purpose is a key concept in privacy policies. Based on the purpose framework developed in our earlier work [11] we present an access control model for a *workflow-based information system* in which a *workflows reference monitor* ("WfRM") enforces purpose-based policies. We use a generic access control policy language and show how it can be connected to the *purpose modal logic language* ("PML") to link purpose constraints to access control rules and how such policies can be enforced. We also present a simple implementation of such a reference monitor based on extending *eXtensible Access Control Markup Language*("XACML"), a commonly used access control open standard.

Keywords: Purpose, Privacy, Purpose-Based Policies, Workflow Reference Monitor, XACML

Contents

1

Introduction

Purpose of use is one of the core concepts in privacy policies and is considered in most privacy laws and regulations. Informally, purpose refers to the user's intention for using data; for example, an employee in an online retailer may use a customer's home address *for the purpose of shipping an ordered good*, or a physician in a research institute may use some of the data in a patient's health record *for research purposes*. Many privacy policies contain rules and restrictions about purpose of data access; for example, an online customer may like to allow access to her email address only for the purpose of sending *order confirmation* and *billing information*, and not for *marketing*, or, the government may decide to prohibit using an applicants' ethnic background information for the purpose of *making hiring decisions*. A *purpose-based privacy policy* is a set of such rules that based on the purpose of access stipulate whether or not it should be allowed.

Enforcing such policies requires a *purpose-based access control* system in which purpose of access is considered a factor in deciding whether or not access should be allowed. For example, the policy may allow an online store employee to use a customer's email address for the purpose of *sending the bill*, but prohibit access to the same data and by the same person/role, for other purpose such as *email marketing*. Traditional access control systems are unable to enforce such rules.

In our earlier work [11] we developed formal semantics for *purpose* based on the situation of an action in a plan of inter-related actions, and discussed some of its important properties, and developed a language, which we call the *purpose modal logic language* ("PML") for formally expressing purpose constraints. In this report, we focus on purpose-based policies and propose a model for their enforcement in a *workflow-based information system*.

The rest of this report is organized as follows: Chapter 2 briefly introduces a workflow-based information system and Chapter 3 discusses some common forms of purpose-based policies with examples.

Chapter 4 presents an abstract model of a *workflow reference monitor* ("WfRM"), a reference monitor for enforcing access control policies in a workflow system. To avoid committing to any specific access control system or language, we introduce a very general and expressive language based on predicate logic, the *access control policy language* ("ACPL") which encompasses most common access control languages. The main intention is to show how purpose constraints (modelled as PML formulas) can be tied to broader access control policy rule (modelled in the ACPL). We show that by modelling the purpose constraint as special predicates in the ACPL, the two languages can be linked and the power of modelling purpose constraints can be added to the ACPL. Moreover, we discuss the abstract enforcement mechanism for policies modelled in ACPL and how this enforcement mechanism should interact with the PML by invoking the purpose model-checker whenever purpose constraints are encountered.

Since due to the powerful nature of the ACPL, the enforcement problem in its general form is undecidable, in Chapter 5, we discuss how by restricting the policy forms and some other limiting assumptions, a simple WfRM can be implemented based on an open standard access control system, *eXtensible Access Control Markup Language* ("XACML") [17]. Finally, Chapter 6 discusses an extension to the PML to accommodate parametrized purposes.

2

Workflow-Basede Information System

A workflow-based information system is a data processing system in which all authorized activities are defined in the form of workflows and no isolated or arbitrary access to data, outside the context of workflows is permitted. Users interact with such systems by either defining new workflows, starting an instance of an already defined workflow, or executing some task in an ongoing workflow (in the form of users' *work lists*). The core of a workflow-based system is the *workflow management system* ("WfMS") which manages definition and execution of all workflows [16]. Because of its crucial role in overseeing all workflow activities, the WfMS is naturally the most suitable place to monitor access to resources, so, a *workflow reference monitor* ("WfRM") is implemented as part of the WfMS.

Figure 2.1 shows the components and their interactions in a workflow-based information system.

The *data repository* is the backend database that stores all data. The *policy database* stores all applicable policies to data processing, ranging from general overarching policies governing the use of all data, to data-specific policies that apply only to specific data.

The *identity service* is in charge of authenticating users as well as storing and maintaining their attributes such as *roles*. It has a query interface for providing the attributes of a particular user to other components that might need them and is trusted in its assertions. To keep our focus on the specifics of our framework we do not consider the issue of authentication and assume that the identity of all users is verified using suitable technologies.

The *policy authoring interface* enables policy makers to formulate the policies governing the use of data. Different authorities can make policies, ranging from overarching jurisdictional policies to personal preference by data subjects (such as consent policies applied to health records authored by the patients or their delegates). The question of who has the authority to make policies about a given resource and to what extent is decided by *administrative policies* which are beyond the scope of our system; we simply assume that all the trusted policies applicable to the use of resources are stored in the policy database.

Depending on the type of the system and policy authors' level of expertise, a suitable graphical or textual interface is used to design policies, but the eventual output of this component is in the form of a standard access control language, such as XACML, and the purpose constraints must be formulated in the form of a PML formula.

The workflows are defined by business process experts and are submitted to the WfMS. The WfMS may verify the workflow definitions from different aspects including compliance to access control policies. Approved workflows are stored in *workflow definitions catalogue*.

The policies and the workflow definitions use a common vocabulary which is defined and stored in the *vocabulary database*. The vocabulary is defined by domain experts and administrators of the system and can be based on international or business standards where applicable.

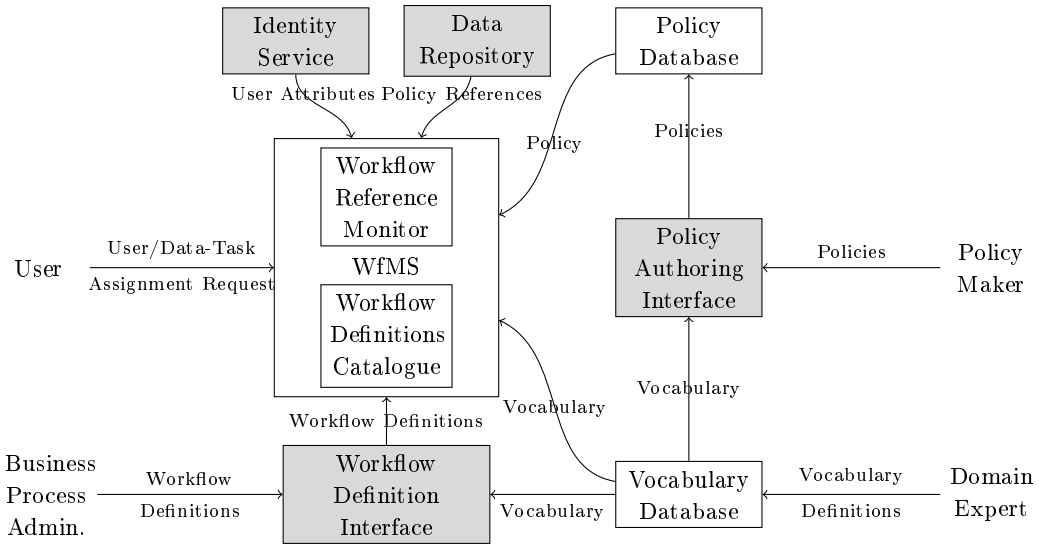

Figure 2.1: System view of a workflow-based system. The greyed-out components are not discussed in our work.

2.1 From a Plan of Actions to a Workflow

Although plans correspond to workflows, they only model the structure of their tasks and an actual workflow often includes many other features. In this section, we discuss some of these additional features that are important in our work.

A *workflow* also known as a *workflow definition* includes the following components:

- The network structure of the workflow tasks which is described in the form of a *hierarchy net* ("HN").
- The mapping that assigns semantic terms from a purpose vocabulary to each transition based on the type of processing performed by the transition.
- A set of workflow parameters such as the definer of the workflow and its administrator. These may be place holders for the actual values which will be specified for an actual instance of the workflow.
- A set of parameters for each task, such as its actor, its data, its resources, and its location. These may be place holders for the actual values which will be specified for an actual instance of the workflow.
- A set of policies that state various constrains about the workflow or task parameters such as separation of duties, role-based restrictions, and temporal/spatial constraints. There are various types of constraints that result from business or access control requirements and have been studied extensively by the research community; for example, see [2] and [4].

A transition with the above additional features is a *task*. An actual running instance of a workflow is called a *workflow instance*. Likewise, a *task instance* is an actual running instance of a task in a workflow instance to which all the parameters are assigned. In Chapter 4 we will give a more formal representation of these objects.

2.2 Workflow Lifecycle

All the business activities in a workflow-based information system are conducted via workflows. In this section we review the event flow for definition and execution of workflows. In Chapter 4 we will discuss the formal details of different access control verifications necessary in this lifecycle.

2.2.1 Workflow Definition

Workflows are defined by business experts to specify authorized and standard ways of doing activities in a system. Workflows are defined using a *workflow definition interface* and is submitted to the WfMS via a *workflow definition request*. If the request is authorized, the WfMS assigns a unique identifier to the workflow and stores it in the *workflow definition catalogue*.

2.2.2 Workflow Instantiation

A user with suitable privileges can request to create a new instance of a workflow that is already defined and stored in the workflow definition catalogue. This request includes the actual values assigned to the parameters of the workflow.

If the request is deemed authorized, a new instance of the workflow is created by assigning an identity and creating the run-time context for keeping track of the state of the workflow according to the operational semantics. The state of a newly instantiated workflow is set to the initial marking in which its source transition is enabled.

2.2.3 Task Instantiation

In order to run a task in a workflow it must be *instantiated*. A request to instantiate a task in an existing workflow instance includes the values assigned to the task parameters. If the request is deemed authorized, the WfMS creates a run-time context for it and puts it in the assigned user's *work list* to be scheduled for execution. Such work lists might be arranged explicitly as a standard list of work items or be integrated as part of a higher-level application [1]. Note the actual instantiation can take place only once the task is enabled according to the operational semantics.

Some workflow systems assume a *static workflow execution model* in which assigning values to all task parameters is performed at workflow instantiation time. A more flexible model, called *dynamic workflow execution model* allows such assignments to be delayed until the task is ready to be executed [5]. This is a more suitable model for larger or cross-organizational workflows in which parameters for future tasks cannot be assigned too long in advance. Also, in case of loops, different iterations of a task may require assignment of different parameters. For example, since a medical care workflow spans a long period of time and may also include some repetitive routines, it is usually not possible to decide the identity of all the nurses assigned to a patient's care process at the time of admission to the hospital.

2.2.4 Task Navigation and Workflow Completion

Once a task is complete, the instance is destroyed, and based on the operational semantics of the workflow, the next ready tasks (i.e. those that are enabled and their parameters are assigned) are enabled. This continues until the workflow instance reaches its final marking after the execution of the final task when the workflow instance is destroyed and cleaned up from the WfMS.

3

Purpose-Based Policies

We defined purpose constraint as a restriction about the purposes of access which is modelled as a formula in the PML. Traditional access control policies, however, are far broader and include rules about *actions*, *subjects* of access, the accessed *resource* and the context in which the access takes place (a.k.a the *environment*) [17]. Such rules can be modelled in an *access control policy language*. Purpose-based policies add *purpose* as an additional decision factor to such rules so that the policy can additionally specify constraints about the purposes of access. In other words, purpose-based policies are formed by linking a purpose constraint to a traditional access control policy.

There are various languages for modelling access control rules and a thorough discussion of the formal semantics of access control policy languages is beyond the scope of our work. In order avoiding committing to any specific access control policy language, we rely on a very general language based on predicate logic which can encompass most existing access control policy languages. We call this generic language the *access control policy language* ("ACPL"). In this chapter, we only use the ACPL and informally explain the meaning of the predicates and functions used in the policies. The informal meaning of a policy expressed in ACPL is that it must not be violated, i.e. must not become equivalent to a contradiction, at any state of the system. Detailed discussions, including the enforcement model for this language, the handling process for access control requests, and interactions with the purpose predicates will be given in Chapter 4.

We assume a special family of predicates exists in ACPL in the general form of $\texttt{Purpose}_{\psi}(\texttt{T})$ in which ψ is a purpose constraint belonging to the PML, and represent whether task `T` satisfies the purpose constraint ψ. This predicate is essentially the connection point between the purpose language and the general access control language.

In the remainder of this chapter, we discuss some common forms of purpose-based policies, based on different ways in which purpose constraints (discussed in [11]) can be tied to access control rules to form general purpose-based access control policies.

3.1 Action-Centric Policies

Action-centric policies are the result of tying a purpose constraint to a specific (type of) action. The general form of such policies is to require a certain purpose constraint to hold as a condition for allowing an action.

As an example consider a policy that prohibits the purpose of *research* for any task instance that is happening *remotely* (i.e. over the Internet):

$$\forall^{TASKI} t\ \texttt{Remote}(t) \rightarrow \texttt{Purpose}_{\psi_1'}(\texttt{Task-Type-Of}(t))$$

in which $\psi_1' \equiv \neg\langle\mathbf{A}\rangle research \wedge \neg\langle\mathbb{F}\rangle research$. The predicate `Remote` specifies whether a task instance is being invoked remotely. The function `Task-Type-Of` returns the task type of a task instance. The policy essentially says that if the task instance is remote, the task must satisfy ψ_1'.

3.2 Subject-Centric Policies

A subject-centric purpose-based policy expresses some constraints based on the subject of access. A black/white list of unauthorized/authorized purposes for roles is a simple form of this type of policy. For example, a hospital's policy may stipulate that the *admin assistant* role cannot perform any actions for the purpose (type A) of *treatment* which can be formulated as follows:

$$\forall^{TASKI} t \ \texttt{Admin-Assistant}(\texttt{User-Of}(t)) \rightarrow \texttt{Purpose}_{\psi'_2}(\texttt{Task-Type-Of}(t))$$

in which $\psi'_2 \equiv \neg\langle\mathbf{A}\rangle$*treatment*. The function `User-Of` returns the user assigned as an actor to a task instance and the predicate `Admin-Assistant` specifies whether a user belongs to the *admin assistant* role.

More complex policies can be expressed using other attributes, for example, a policy may require that in order to access data for the purpose of *genetic analysis* the actor should have the *physician* role and at least 5 years of experience in the practice:

$$\forall^{TASKI} t \ \texttt{Purpose}_{\psi'_3}(\texttt{Task-Type-Of}(t)) \rightarrow (\texttt{Physician}(\texttt{User-Of}(t)) \wedge (\texttt{Experience}(\texttt{User-Of}(t)) \texttt{>= 5}))$$

in which $\psi'_3 \equiv \langle\mathbf{A}\rangle$*genetic-analysis* $\vee \langle\mathbf{F}\rangle$*genetic-analysis*. The predicate `Physician` specifies whether a user belongs to the role `Physician` and the function `Experience` returns the number of years of experience a user has in her or his position.

3.3 Data-Centric Policies

Data-centric policies are composed of linking some purpose constraints to a certain data resource, i.e. whenever that data resource is accessed, certain purpose constrains must hold. This is a very common approach for purpose-based policies and has been adopted by many models and standards [18, 3]. A black/white list of unauthorized/authorized purposes (a.k.a *intended purposes* [3]) attached to a data resource is the simplest form of this kind of policies. For example, a patient consent for Alice's record may require that the medical record shall not be used for the purpose of *marketing*:

$$\forall^{TASKI} t \ \texttt{Owner-Of}(\texttt{Data-Of}(t)) = \texttt{Alice} \rightarrow \texttt{Purpose}_{\psi'_4}(\texttt{Task-Type-Of}(t))$$

in which $\psi'_4 \equiv \neg\langle\mathbf{A}\rangle$*marketing* $\wedge \neg\langle\mathbb{F}\rangle$*marketing*. The function `Data-Of` returns the identifier of a data resource assigned to a task instance, and the function `Owner-Of` returns the identifier of the owner of the data resource. The policy basically says if Alice's record is assigned to a task instance, the task must satisfy ψ'_4.

More complex policies can be formed by considering other attributes of the data resource, for example, a jurisdictional policy may stipulate that if the data labelled with *reproductive health* it must not be accessed for *research*:

$$\forall^{TASKI} t \ \texttt{Rep-Health}(\texttt{Data-Of}(t)) \rightarrow \texttt{Purpose}_{\psi'_5}(\texttt{Task-Type-Of}(t))$$

in which $\psi'_5 \equiv \neg\langle\mathbf{A}\rangle$*research* $\wedge \neg\langle\mathbb{F}\rangle$*research*. The predicate `Rep-Health` specifies whether a data item is labelled with *reproductive health*.

Note that data is not the only type of resource in the system and other resources may also have their own resource-specific policies. For example, the policy for using a printer in an office (`PRINTER-1`) might require the purpose of *billing confirmation* for printing, to prevent using the printer for other purposes.

$$\forall^{TASKI} t \ \texttt{Assigned-Resource}(t) = \texttt{PRINTER-1} \rightarrow \texttt{Purpose}_{\psi'_6}(\texttt{Task-Type-Of}(t))$$

in which $\psi'_6 \equiv \langle\mathbf{A}\rangle$*billing-confirmation* $\vee \langle\mathbf{F}\rangle$*billing-confirmation*. This policy says the if `PRINTER-1` is assigned as a resource to any task instance, that task must comply with the purpose constraint ψ'_6.

3.4 Environment-Centric Policies

Likewise, environment-centric policies tie a purpose constraint to some environmental attributes such as time or location. For example, a policy may require that the purpose of *marketing* is not authorized for access outside business hours, or the purpose of *public health* is only authorized when access is requested within a particular country or province:

$$\forall^{TASKI} t \ \texttt{Purpose}_{\psi_7'}(\texttt{Task-Type-Of}(t)) \rightarrow \texttt{Local}(\texttt{Location-Of}(t))$$

in which $\psi_7' \equiv \langle \mathbf{A} \rangle$*public health* $\vee \langle \mathbf{F} \rangle$*public health*. The function `Location-Of` returns the location identifier for a task instance and the predicate `Local` specifies whether a location identifier is considered local to the province or the country.

3.5 Compound Policies

Complex policies can be formed by combining action-, subject-, data- and environment-centric rules. For example, a subject-data-environment-based policy may restrict purpose of *research* to users with the role of *physician* and only on *anonymized* data and within the location of the organization premises. Some privacy policy languages such as the Enterprise Privacy Authorization Language ("EPAL") [6] model such rules in the form of quadruples that specify authorized or unauthorized combination of subjects, actions, resources and purposes. The model we have followed here is much more general.

However, modelling policies in the subject-, or data-centric form, facilitates their storage/retrieval and finding the corresponding policies to an access request. For example, data-centric policies can be checked easier since the applicable policies for a data item can be fetched and checked whenever the data item is about to be used. We discuss this further in the next chapter.

4

Access Control Model

In this chapter we discuss enforcement of purpose-based policies by an access control system. Since purpose of an action depends on its situation in the workflow context as defined by the purpose framework in [11], recognition of purposes require a workflow-based information system. We assume a *workflow-based information system*, as introduced in Chapter 2, is in place.

In the rest of this chapter we discuss a very general abstract model of WfRM and show how it should work to enforce access control policies. Aligned with this general model, we discuss the details of a sample WfRM. To avoid committing to details of any specific access control language or system, we use a very general predicate logic which we call *access control policy language* ("ACPL"). This language is very general and therefore can represent most existing access control policy languages. Access control policies are modelled as formulas in ACPL and purpose constraints are modelled using the *purpose modal logic language* ("PML"). We discuss how the two languages can be connected by encapsulating PML formulas as predicates in ACPL, and thus, how purpose constraints can be tied to different types of access control policies. We also discuss in detail how the two languages interact in evaluating the policies, i.e. how purpose model-checking algorithm is used to evaluate purpose constraints in the course of evaluating ACPL formulas. We discuss the semantics for three essential operations in a workflow system, namely, *workflow definition*, *workflow instantiation*, and *task instantiation* together with examples for each. We also present numerous examples to demonstrate the above mechanisms.

Following this abstract discussion, in Chapter 5 we will show how by making some restrictive assumptions about the policy form and some other limitations, the WfRM can be implemented using an actual access control product, the *eXtensible Access Control Markup Language* ("XACML") [17].

4.1 Abstract Model of the WfRM

We define the WfRM as a pair $\langle \mathcal{G}, \Sigma \rangle$ in which $\mathcal{G}$ is the *global schema* and Σ is a *deterministic automaton*. The *global schema* is a set that represents the static part of the system which remains constant and is not subject to change by any operation. The automaton represents the dynamic part of the system which interacts with the users and the environment and changes accordingly.

Definition 1 (Workflow Reference Monitor Automaton)**.** *The* WfRM automaton *is a quadruple* $\langle S, Q, \delta, s_0 \rangle$ *in which:*

- S *is the set of* states*,*
- Q *is the set of* requests*,*
- $\delta : S \times Q \mapsto S$ *is a partial function called the* transition function*, and*
- $s_0 \in S$ *is the* initial state*.*

The *system state* is a triple $\langle \mathcal{A}^S, \Phi, \mathcal{C} \rangle$ in which $\mathcal{A}^S$ is the set of *state assertions* which represents all

the current facts about the system at the current state, Φ represents the set [1] of applicable *policies* at the current state including purpose-based policies, and $\mathcal{C}$ contains any other miscellaneous state information that the system needs to keep track of. Since we do not consider $\mathcal{C}$ in our discussion, and in order to keep the notation brief, we simply show a state as $\langle \mathcal{A}^G, \Phi \rangle$.

The global schema $\mathcal{G}$ can be modelled in the form of a pair $\langle \mathcal{A}^G, G \rangle$, in which $\mathcal{A}^G$ is a set of *global assertions* and G encapsulates anything else that is part of the global schema. Both *global* and *state* assertions, as well as *policies* are expressed using formulas in a suitable language which includes the logical connectives such as negation ($\neg$) and conjunction ($\wedge$) in their conventional meaning. As a condition for *integrity*, the assertions and the policies must be individually consistent.

Definition 2 (Integrity Condition)**.** *A system state $s = \langle \mathcal{A}^S, \Phi \rangle$ abides by the* integrity condition *if and only if there are no contradictions in the set of assertions or the set of policies, i.e. $\bigwedge_{a_i \in \mathcal{A}^S \cup \mathcal{A}^G} a_i$ and $\bigwedge_{\phi_i \in \Phi} \phi_i$ are both* satisfiable *formulas*[2]*:*

The goal of a reference monitor is to ensure that, at all times, the facts of the system (represented by the *assertions*) comply with the applicable rules (represented by the *policies*). We call such desired states of the system *compliant states*.

Definition 3 (Compliant State)**.** *A system state $s = \langle \mathcal{A}^S, \Phi \rangle$ is a* compliant state *if and only if the following formula is* satisfiable*:*

$$\Gamma(s) \equiv \left(\bigwedge_{a_i \in \mathcal{A}^S \cup \mathcal{A}^S} a_i \right) \wedge \left(\bigwedge_{\phi_i \in \Phi} \phi_i \right) \tag{4.1.1}$$

Note that based on this definition, every *compliant state* also abides by the integrity condition. For brevity, in the rest of this chapter, whenever we drop the superscript from $\mathcal{A}$ at a given state, we mean the union of the global and state assertions: $\mathcal{A} = \mathcal{A}^S \cup \mathcal{A}^G$.

The system starts in an *initial state* and users and the environment interact with it by sending different *requests* to change the facts ($\mathcal{A}^S$) and/or applicable policies (Φ), and thereby change the *state* of the system. The semantics of each request is defined by how it wants $\mathcal{A}^S$ and Φ to be updated.

Definition 4 (Request)**.** *A request $r : S \mapsto S$ is a mapping over the set of all possible states of the system which defines how it would like to update a given state s to a new state $r(s)$.*

The WfRM must ensure that the system is always in a *compliant state*. If the system starts with a *compliant state* and only transitions to a *compliant state* are allowed, by induction, the system is in a compliant state at all times. So, we require that the *initial state* $s_0 = \langle \mathcal{A}_0^S, \Phi_0 \rangle$ be a compliant state and we define the transition function such that it only defines transitions to *compliant states*, i.e., the following condition must be met by δ:

$$\langle s, r, r(s) \rangle \in \delta \Rightarrow r(s) \text{ is a compliant state.}$$

4.2 A Sample WfRM

Based on the general abstract model defined in the previous section, in this section we present a more concrete example of a WfRM. We define the global schema, and some of the essential requests and discuss how they are processed. For formulating $\mathcal{A}$ and Φ, we use a very general *access control policy language* ("ACPL") based on *many-sorted predicate logic* with equality. A brief introduction to this language is given in Appendix A.

[1] The reason we have modelled policies and assertions as *sets* rather than single formulas in conjunctive form is to have clear semantics for *adding* and *removing* policies and assertions to/from these components of the system.

[2] The standard definition of *satisfiability* is that there exists a *model* which satisfies the formula. See the definition of *model* in Section A for an example.

Table 4.1 gives a summary of some of the types, constants, functions, and predicates we use throughout this section in our examples.

All access control rules are modelled using the ACPL. Any policy that includes some purpose constraints, uses a special family of predicates $\texttt{Purpose}_\psi$ which specify whether a task satisfies the formula ψ belonging to the PML. Note that purpose properties are static attributes of tasks and are the same across all instances. For example, a marketing task is always assigned the *marketing* label regardless of the task instance parameters such as the assigned data or actor. Thus, the purpose predicates only depend on the task and workflow structure and do not depend on a particular instance of a task and its parameters. In other words, that is the reason we define the signature of the purpose predicates to task an input of type *task*, and not *task instance* or any other parameters. This works well unless the purposes take parameter themselves, and we need, for example, to distinguish the *purpose of treatment of patient A* from the *purpose of treatment of patient B*. We discuss this extension in Chapter 6.

Purpose predicates essentially, connect the PML to the ACPL by encapsulating the PML formulas as predicates and making them available to be used in any access control policy. Note that the ACPL is very expressive and can model any general access control policy which depends on various types of parameters such as task actors, task data, user roles, time and location, *etc.* Meanwhile, the PML is a limited language only for modelling purpose constraints. Particularly, the PML does not support variables and cannot be used for modelling rules that are sensitive to dynamic properties of the workflow such as the workflow parameters. Since the ACPL is capable of modelling such dynamic properties, connecting the PML to the ACPL via purpose predicates basically brings the power of adding purpose constraints to any complex access control policy expressible using the ACPL.

Details of this predicate family and its evaluation are discussed in Section 4.2. As a notation convention, we use ψ for the formulas belonging to the purpose language and ϕ for the ACPL formulas.

The set of *assertions* ($\mathcal{A}$) in this system contains the facts about the system entities and their relationships. For simplicity, we require the assertions to be *ground formulas* i.e. they contain no quantifiers and no variables. Entities in the system are represented as constants in the language and various functions and predicates are used to formulate their attributes and relationships.

As shown in Table 4.1, the entities we consider are: *workflow definitions*, *workflow instances*, *tasks*, *task instances*, *users*, and *data items*. Some examples of environment assertions are the *workflow type* of a workflow instance, *workflow instance* to which a task instance belongs, and whether a user qualifies for a certain *role* [3].

For simplicity, in this system we assume that the policies are *closed formulas* [4] with only universal quantifiers, i.e. of the form: $\forall t_1 \cdots \forall t_n \phi$. As pointed out by [8], this form covers most of the existing access control policy languages and typical policy rules. Some examples of policies that we consider in this system are *overarching policies*, *data-centric policies* like *consent directives*, and workflow- and task-specific *constraints*.

Global Schema

The global schema $\mathcal{G}$ includes all those components that are not part of the state of the WfRM, i.e. do not change as the state of the WfRM changes. Recalling that $\mathcal{G} = \langle \mathcal{A}^G, G \rangle$ in which $\mathcal{A}^G$ is the set of global assertions and G includes other global properties of the system, we discuss the components of $\mathcal{G}$ in this section [5].

Users and Roles We assume *Users* represents the set of all users, and *Roles* represents the set of all roles in the system. The function $URA : Users \mapsto 2^{Roles}$ assigns to each user the roles for which she or he qualifies.

[3] In order to create the assertions about some attributes, the system state needs to keep a mapping of constants in ACPL to concrete objects they represent, such as *workflows*, *tasks*, their *instances*, *users*, and *data*. For brevity we assume this is implicit in the miscellaneous component of the state ($\mathcal{C}$) and do not discuss it any further.

[4] A *closed formula*, also known as a *sentence*, is a formula with no free variable, i.e. a formula which has no variables at all, or in which all variables appear within the scope of a quantifier.

[5] Note that in the interest of simplicity, we use the same symbol for equality in the ACPL and in the language we use to define the model.

For each role, we assume a predicate with the signature *USER* which denotes whether a user qualifies for that role.

Data and Consent We assume *Data-Items* represents the set of all data items in the system. Moreover, *Consent*, a mapping from *Data-Items* to the set of all closed formulas in ACPL assigns a *consent policy* to each data item. This policy has the form:

$$\forall^{TASKI}\ t\ (\texttt{Data-Of}(t) = \texttt{D}_i \rightarrow \phi)$$

which stipulates that ϕ must not be violated if the particular data item is being used in some task. Normally, t appears as a free variable in ϕ.

Note that data items may have other attributes such as *categories*, *sensitivity level*, *assurance level*, *etc.* Although to keep the language simple, we do not model such attributes here, modelling them is straightforwardly similar to modelling of roles for users.

Task and Workflow Parameters We assume Π^w and Π^t are the set of symbols that represent respectively all the workflow- and task-specific parameters in the system such as workflow instantiator, task actor, and task data.

For each parameter $\pi \in \Pi^w$ we define a corresponding function $\texttt{WF-Parameter}_\pi$ with the input of type *workflow instance* (*WFI*) and a suitable output type. Similarly, for each parameter $\pi \in \Pi^t$, we define a corresponding function $\texttt{Task-Parameter}_\pi$ with the input of type *task instance* (*TASKI*) and a suitable output type [6]. Two of the most important task parameters are `User-Of` and `Data-Of` which return the user and the data item assigned to a task instance.

Task Attributes Aside from parameters, each task may also have some attributes. Corresponding to each attribute a, we define a predicate $\texttt{Attr}_a$ which denotes whether the attribute holds for a task. The input signature for these predicates includes at least one parameter of type *TASK* and depending on the attribute it may take other parameters as well.

The only attributes that we consider here are the semantic attributes that specify the types of processing that the task performs, i.e. the purpose labels. We assume the set $\mathcal{V}$, the *purpose vocabulary* consists of all the semantic attributes of tasks which are also the purpose names. We assume that the predicates corresponding to the purpose vocabulary only have one input of type *TASK* which specify whether the particular attribute holds at a given task. However, as an extension, we consider the case where purpose attributes may take other parameters as well, as discussed in Chapter 6.

Purpose Constraints For each formula ψ belonging to the PML, we define a corresponding purpose predicate shown as $\texttt{Purpose}_\psi$ with the input signature *TASK* which indicates whether a task satisfies the formula ψ in the context of the hierarchy net it belongs to and the *labelling function* that is derived from the task attributes as will be discussed in Section 4.2.

Global Assertions On the other hand, $\mathcal{A}^G$ contains all the global assertions about the system. In this sample system we assume the following are in $\mathcal{A}^G$: For each role *Role* and each user *U*, the corresponding assertion either in the form $\neg\texttt{Role(U)}$ or `Role(U)`, depending whether the user qualifies for the role. `U` is the corresponding constant to *U* and `Role` is the corresponding predicate to *Role*.

[6] Note that for simplicity, we assume that all tasks/workflows share the same set of parameters; in other words, a workflow parameter is meaningful for all workflow instances, and a task parameter is meaningful for all task instances. For example, the parameter `User-Of` exists for all tasks. However, in practice, this may not be the case; for example, there might be tasks that need no users. By using special objects such as `auto` or `null` in the assignment of parameters to such tasks/workflows, this assumption can be preserved.

Symbol	Signature	Description
USER	*type*	The type representing users.
DATA	*type*	The type representing data.
WF	*type*	The type representing workflow definitions.
WFI	*type*	The type representing workflow instances.
TASK	*type*	The type representing tasks.
TASKI	*type*	The type representing task instances.
$\mathtt{U}_0, ..., \mathtt{U}_n$	*USER*	Constants representing different users.
$\mathtt{D}_0, ..., \mathtt{D}_n$	*DATA*	Constants representing different data items.
$\mathtt{W}_0, ..., \mathtt{W}_n$	*WF*	Constants representing different workflow definitions.
$\mathtt{N}_0, ..., \mathtt{N}_n$	*WFI*	Constants representing different workflow instances.
$\mathtt{T}_0, ..., \mathtt{T}_n$	*TASK*	Constants representing different tasks.
$\mathtt{K}_0, ..., \mathtt{K}_n$	*TASKI*	Constants representing different task instances.
`WF-Type-Of`	(*WFI*, *WF*)	Function that returns the workflow type of a workflow instance.
`Task-Type-Of`	(*TASKI*, *TASK*)	Function that returns the task type of a task instance.
`WF-Of`	(*TASK*, *WF*)	Function that returns the workflow definition to which a task belongs.
`WFI-Of`	(*TASKI*, *WFI*)	Function that returns the workflow instance to which a task instance belongs.
$\mathtt{Purpose}_{\psi}$	(*TASK*)	Predicate that indicates whether a task satisfies a modal logic formula ψ belonging to the purpose language. Evaluation of this predicate is discussed in Section 4.2.
$\mathtt{WF\text{-}Parameter}_{\pi}$	(*WFI*, *type*)	Function that returns the value of the workflow parameter π for a workflow instance. Some specific examples are given below.
$\mathtt{Task\text{-}Parameter}_{\pi}$	(*TASKI*, *type*)	Function that returns the value of the workflow parameter π for a task instance. Some specific examples are given below.
$\mathtt{Attr}_{a}$	(*TASKI*)	Predicates indicating whether a task holds the attribute a.
`Definer-Of`	(*WF*, *USER*)	Function that returns the user who defined the workflow.
`WF-Admin-Of`	(*WFI*, *USER*)	Function that returns the user who instantiated a workflow instance.
`Task-Admin-Of`	(*TASKI*, *USER*)	Function that returns the user who instantiated a task instance.
`User-Of`	(*TASKI*, *USER*)	Function that returns the user assigned to perform a task instance (Note that this presumes a single user is assigned to each task instance).
`Data-Of`	(*TASKI*, *DATA*)	Function that returns the data assigned to a task instance (Note that this presumes a single data item is assigned to each task instance).
`Jun-Res`	(*USER*)	Predicate indicating whether a user qualifies as *junior researcher*.
`Sen-Res`	(*USER*)	Predicate indicating whether a user qualifies as *senior researcher*.
`Admin`	(*USER*)	Predicate indicating whether a user qualifies as *administrator*.

Table 4.1: ACPL components used in our examples.

Requests

In this section, we discuss the details for three main types of requests in the workflow system. As a notational convention, we show concrete objects with capital-letter italic font (like T_1), constants in the ACPL with capital typewriter font (like $\mathtt{T}_1$), and ACPL variables with small-letter italic font (like t). We also use the same names to denote correspondence between the constants and the concrete objects; for example, $\mathtt{T}_1$ is the corresponding constant to the concrete object T_1.

Workflow Definition A workflow definition request includes the following components:

- The identity of the user who defines the workflow, U_i.
- A workflow definition identifier, W_i.
- Workflow definition details, which is a pair $\langle \mathcal{H}, M \rangle$ as follows:
  - $\mathcal{H}$ is the hierarchy net which defines the workflow structure and run-time behaviour, and
  - $M \subseteq \mathcal{T}_{\mathcal{H}} \times \mathcal{V}$, the mapping of each task to the semantic labels from the purpose vocabulary.
- ϕ_i is a formula belonging to the ACPL which models all the workflow- and task-specific *constraints*.

Assuming $r(s) = \langle \mathcal{A}', \Phi' \rangle$, the semantics of this request are defined as follows:

- Let $\Phi' = \Phi \cup \{\phi_i\}$.
- Let $\mathcal{A}' = \mathcal{A} \cup \{\mathcal{A}_i\}$ where $\mathcal{A}_i$ is the conjunction of all the assertions pertaining to the new workflow definition, such as:
  - The assertion about the identity of the definer of the workflow, i.e. $\mathtt{Definer\text{-}Of}(\mathtt{W}_i) = \mathtt{U}_i$ where $\mathtt{U}_i$ and $\mathtt{W}_i$ are constants representing W_i and U_i.
  - For every task in the workflow, an assertion about the workflow to which it belongs, i.e. $\mathtt{WF\text{-}Of}(\mathtt{T}_i) = \mathtt{W}_i$ for every $T_i \in \mathcal{T}_{\mathcal{H}}$ (where $\mathtt{T}_i$ is a constant representing T_i).
  - The assertion of purpose attributes for all the tasks of the new workflow for every $T_i \in \mathcal{T}_{\mathcal{H}}$ and every $v \in \mathcal{V}$, i.e. $\mathtt{Attr}_v(\mathtt{T}_i)$ if and only if $v \in L(T_i)$ and $\neg\mathtt{Attr}_v(\mathtt{T}_i)$ if and only if $v \notin M(T_i)$.

Workflow Instantiation A workflow instantiation request includes the following components:

- The identity of the user who makes the instantiation request U_i.
- A workflow definition identifier W_i of the workflow to be instantiated.
- The value for all workflow parameters, i.e. a mapping from Π^w to $\{V_1, \cdots, V_n\}$ where each V_i is of the suitable type and represents the value of a parameter.

Assuming $r(s) = \langle \mathcal{A}', \Phi' \rangle$, the semantics of this request are defined as follows: $\mathcal{A}' = \mathcal{A} \cup \{\mathcal{A}_i\}$ where $\mathcal{A}_i$ is the conjunction of all the assertions pertaining to the new workflow instantiation, such as the following. The workflow instance is given a fresh constant $\mathtt{N}_i$ as identifier.

- The assertion about the type of workflow instance, i.e. $\mathtt{WF\text{-}Type\text{-}Of}(\mathtt{N}_i) = \mathtt{W}_i$.
- The assertion about the user who creates the workflow instance, i.e. $\mathtt{WF\text{-}Admin\text{-}Of}(\mathtt{N}_i) = \mathtt{U}_i$.
- The assertions about the values assigned to workflow parameters, i.e. $\mathtt{WF\text{-}Parameter}_{\pi}(\mathtt{N}_i) = \mathtt{V}_i$ for each parameter π whose value is given as V_i ($\mathtt{V}_i$ is the corresponding constant to the parameter value V_i). Note that $\mathtt{V}_i$ may be a fresh (e.g. time or location identifier) or an already existing constant (e.g. an existing user identifier).

Task Instantiation A task instantiation request includes the following components:

- The identity of the user who makes the instantiation request, U_i.
- The workflow instance identifier to which the task belongs, W_i.
- The task identifier T_i.
- The value for all task parameters, i.e. a mapping from Π^t to $\{V_1, \cdots, V_n\}$ where each V_i is of the suitable type and represents the value of a parameter.

Assuming $r(s) = \langle \mathcal{A}', \Phi' \rangle$, the semantics of this request are defined as follows: $\mathcal{A}' = \mathcal{A} \cup \{\mathcal{A}_i\}$ where $\mathcal{A}_i$ is the conjunction of all the assertions pertaining to the new task instantiation, such as the following. The task instance is given a fresh constant $\mathtt{K}_i$ as identifier.

- The assertion about the type of the task instance, i.e. $\mathtt{Task\text{-}Type\text{-}Of}(\mathtt{K}_i) = \mathtt{T}_i$.
- The assertion about the workflow instance to which the task instance belongs, i.e. $\mathtt{WFI\text{-}Of}(\mathtt{K}_i) = \mathtt{W}_i$.
- The assertion about the user who creates the task instance $\mathtt{Task\text{-}Admin\text{-}Of}(\mathtt{K}_i)$.
- The assertion about the value assignment to all the task parameters, i.e. $\mathtt{Task\text{-}Parameter}_\pi(\mathtt{N}_i) = \mathtt{V}_i$ for each parameter π whose value is given as V_i (note that we use $\mathtt{V}_i$ as the corresponding constant to the parameter value V_i). Two significant parameters in our examples are represented by functions `User-Of` and `Data-Of`. Note that $\mathtt{V}_i$ may be a fresh (e.g. time or location identifier) or an already existing constant (e.g. an existing user identifier).

Moreover, for the assigned data item to the task (given by the function `Data-Of`), the corresponding consent policy of the data item is added to the set of policies, i.e. $\Phi' = \Phi \cup \{\phi_i\}$ where $\phi_i = \mathit{Consent}(D_i)$, $\mathtt{D}_i = \mathtt{Data\text{-}Of}(\mathtt{K}_i)$, and $\mathtt{D}_i$ is the constant corresponding to D_i.

Other Requests There are various other requests in the system. For example, the actual event of performing a task instance may introduce some assertions about its starting time and location and if the policy is sensitive to such parameters, they need to be checked. Or, finishing a task or conclusion of a workflow instance leads to removing the instance and corresponding assertions from the system state, including the removal of consent policies of the data assigned to those tasks. Since handling these requests are essentially very similar to the ones discussed above, we do not discuss their details here.

Evaluating the Purpose Predicates

Purpose predicates in the ACPL are gateways to the PML with which purpose constraints can be modelled. When it comes to evaluating policies, thus, we need to establish the link which specifies how the purpose model-checking algorithm must be invoked for evaluating purpose predicates, and how this fits in the process of evaluating the rest of ACPL formulas.

In this section we define how purpose predicates of the form $\mathtt{Purpose}_\psi(\tau)$ can be evaluated for a ground term τ. Note that a *ground term* is a term that contains no variable. This mechanism can be used in two ways:

- In a policy analysis system based on deduction, this can be used to replace occurrences of $\mathtt{Purpose}_\psi(\tau)$ with $\top$ or $\bot$ whenever τ is ground.
- In a model checking system like the one we present in Chapter 5, this can be used to compute the interpretation of $\mathtt{Purpose}_\psi$ for a model, i.e. computing the truth value of this predicate for concrete objects. We use this in Section 5.2.3 and present more details.

The main idea is that since the PML only supports atomic propositions, the required knowledge for evaluating purpose constraints modelled in ACPL must be imported to the PML in the form the *labelling function* which is given as an input to the purpose model-checker. This is done by converting ground predicates assertions about task attributes into atomic propositions and assigning them accordingly to the workflow tasks. Once this is done, the truth value for $\texttt{Purpose}_\psi(\texttt{T})$ for a specific task T can be decided by invoking the purpose model-checker. We explain this process below.

In order to keep this discussion concise and more comprehensible, we use the font coding to show the correspondence between constants symbols and the concrete objects to which they correspond.

- Using the $\texttt{W} = \texttt{WF-Of}(\texttt{T})$ parameter assertion, the workflow definition W to which T belongs can be extracted. The actual workflow definition for W in the system includes the definition of the hierarchy net $\mathcal{H}$ which gives the structure of the HN and other tasks belonging to this workflow.

- For each task T_i belonging to $\mathcal{T}_\mathcal{H}$ and for each vocabulary term v belonging to the purpose vocabulary $\mathcal{V}$, we assume an atomic proposition v_p belonging to the purpose language. The truth value of v_p at each task is assigned based on the truth value of the predicate $\texttt{Attr}_v$. In other words the labelling function for the purpose model-checker is built by setting $v_p \in L(T_i)$ if and only if $\texttt{Attr}_v(\texttt{T}_i)$. Note that as discussed in Section 4.2 for each task attribute we add either a positive or negative assertion, thus, after the workflow definition, for each attribute and each task we either have $\texttt{Attr}_v(\texttt{T}_i)$ or $\neg\texttt{Attr}_v(\texttt{T}_i)$.

- According to the result of the purpose model-checking algorithm if $T \in [\![\psi]\!]_{\mathcal{H},L}$ we will have $\texttt{Purpose}_\psi(\texttt{T}) \equiv \top$, otherwise $\texttt{Purpose}_\psi(\texttt{T}) \equiv \bot$. Similarly, for doing ACPL model-checking, if $T \in [\![\psi]\!]_{\mathcal{H},L}$, then $(T, \mathit{true}) \in \texttt{Purpose}_\psi^\mathcal{M}$, otherwise $(T, \mathit{false}) \in \texttt{Purpose}_\psi^\mathcal{M}$ where $\texttt{Purpose}_\psi^\mathcal{M}$ is the interpretation of the predicate $\texttt{Purpose}_\psi$.

Note that the labelling function L created in this process is essentially the same as the mapping M given as part of the workflow definition (see Section 4.2). The reader may wonder why we changed this information into assertions, only to convert them back into the form of a labelling function L. This can in fact be used as a shortcut to simplify the implementation as we will see in Section 5.2.3. However, there are various reasons for making this distinction: First, although M and L are very similar, they are different from the semantics viewpoint: one represents some information about the workflow definition and the other provides the mapping of tasks to atomic propositions in the Kripke model. Thus, even if we do not do the conversion as we did, an implicit conversion is necessary from *vocabulary terms* to *atomic propositions*. Secondly, the mapping M is part of the general knowledge in the system and since we have decided to represent all the knowledge about the system in the form of ACPL assertions, it is necessary to formulate M in the form of assertions in the system state. In other words, since we did not establish a direct connection between the purpose language and the outside world, any information needed by the purpose model-checker need to come from the system assertions.

Moreover, in case purpose attributes have other parameters, such as the data assigned to the task, as we will discuss in Chapter 6, such static mapping will not hold between the vocabulary and the atomic propositions in the purpose language and assigning atomic propositions to tasks is not possible until those parameters are known and the corresponding predicate can be grounded.

Walk-Through Example

Assume Alice, a researcher in the institute, needs to perform a correlation analysis between certain demographic information and immunity to hepatitis. This is an activity defined as workflow W_1 (Figure 4.2). The medical record of a patient identified as D_1 is stored in the database together with the corresponding consent directive. The details of the overarching and consent-directive policies, vocabulary, workflow and labelling function, as well as the event flow of the WfRM in the course of defining, instantiating and running this workflow are discussed in this section.

Overarching Policies The following are the overarching policies of the system:

- Any user defining any workflow must have the role *admin*:

$$\phi_1 \equiv \forall^W w \ \texttt{Admin}(\texttt{Definer-Of}(w)) \tag{4.2.1}$$

- Only the user who instantiated the workflow can instantiate its tasks:

$$\phi_2 \equiv \forall^{TASKI} t \ \texttt{WF-Admin-Of}(\texttt{WFI-Of}(t)) = \texttt{Task-Admin-Of}(t) \tag{4.2.2}$$

- The role *senior researcher* is required for the actor of any activity that is for the purpose of *immunologic procedure* (ip):

$$\phi_3 \equiv \forall^{TASKI} t \ \texttt{Purpose}_{\psi_1}(\texttt{Task-Type-Of}(t)) \rightarrow \texttt{Sen-Res}(\texttt{User-Of}(t)) \tag{4.2.3}$$

  in which $\psi_1 \equiv \langle \mathbf{A} \rangle \texttt{ip} \vee \langle \mathbf{F} \rangle \texttt{ip}$ [7].

Consent Directive The patient's consent is an *order-based* purpose constraint: "access to this record is prohibited for the purpose of *immunologic procedure* (ip), if this purpose is in turn for the purpose of *research* (r)". Note that this does not prohibit other *research* purposes than *immunologic procedure*, for example, the purpose of *correlation of alcohol use and liver cancer* for *research* is still allowed. Also, it does not prohibit access for the purpose of *immunologic procedure* altogether; the record can still be used for that purpose if it is not in turn for *treatment* purposes.

This policy can be formulated as the following; $\texttt{D}_1$ is the constant corresponding to data item D_1:

$$\phi_4 \equiv \forall^{TASKI} t \ \big((\texttt{Data-Of}(t) = \texttt{D}_1) \rightarrow \texttt{Purpose}_{\psi_2}(\texttt{Task-Type-Of}(t))\big) \tag{4.2.4}$$

in which $\psi_2 \equiv \neg\langle \mathbf{A} \rangle(\texttt{ip} \wedge (\langle \mathbf{A} \rangle \texttt{r} \vee \langle \mathbb{F} \rangle \texttt{r})) \wedge \neg\langle \mathbb{F} \rangle(\texttt{ip} \wedge (\langle \mathbf{A} \rangle \texttt{r} \vee \langle \mathbb{F} \rangle \texttt{r}))$.

Vocabulary We assume the standard vocabulary in use is based on SNOMED-CT [14] and NCI Thesaurus [12]. Both vocabularies come in the form of an ontology and include semantic relationships among terms. Here, we only consider the *is-a* relations. Figure 4.1 shows a portion of the vocabulary used in this example with the *is-a* relations.

Workflow Definition The workflow W_1 is shown in Figure 4.2. The workflow definition has two main parts; the hierarchy net and labelling function and the workflow-specific policies.

HN and Labelling It starts by setting up the test instance (T_0) and then simply looping over a number of patients and enrolling each patient in the research procedure (T_1), reading and registering age, gender and ethnicity (T_2, T_3, and T_4) and then proceeding to check and register the immunity of the patient to hepatitis (T_5). When there is enough samples, the workflow performs some mathematical correlation analyses (T_6) and concludes by saving the results (T_7). This workflow is one of the realizations of the activity *testing demographic factors* (T') which is in turn one realization of *autoimmune hepatitis study process* (T'').

The workflow designer assigns terms from the vocabulary to workflow tasks based on their semantics. For example, the task *check hepatitis immunity* is mapped to the atomic proposition *hepatitis immunity test* from SNOMED-CT. The parents of this term in the ontology are also assigned to the task (non-boldfaced labels in Figure 4.2). Since the ontological relations exist in machine-readable form, adding the parents of an assigned label is done automatically by the workflow definition tool. Note that this can be extended so that the language and the model-checking algorithm support the ontology relations natively as discussed in [11].

[7] Note that to avoid complicating the notation and in the interest of simplicity, we use same symbols for logical connectives in both ACPL and the purpose modal-logic language. The difference should be obvious from the context

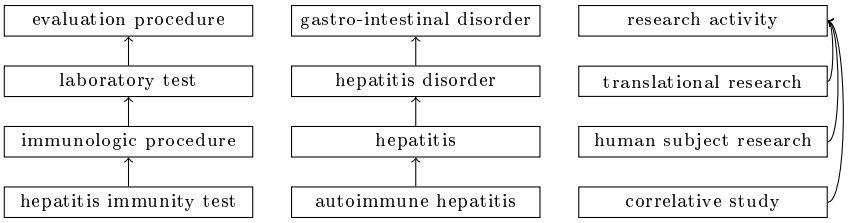

Figure 4.1: Some SNOMED and NCI terms used in our example with their hierarchical relations.

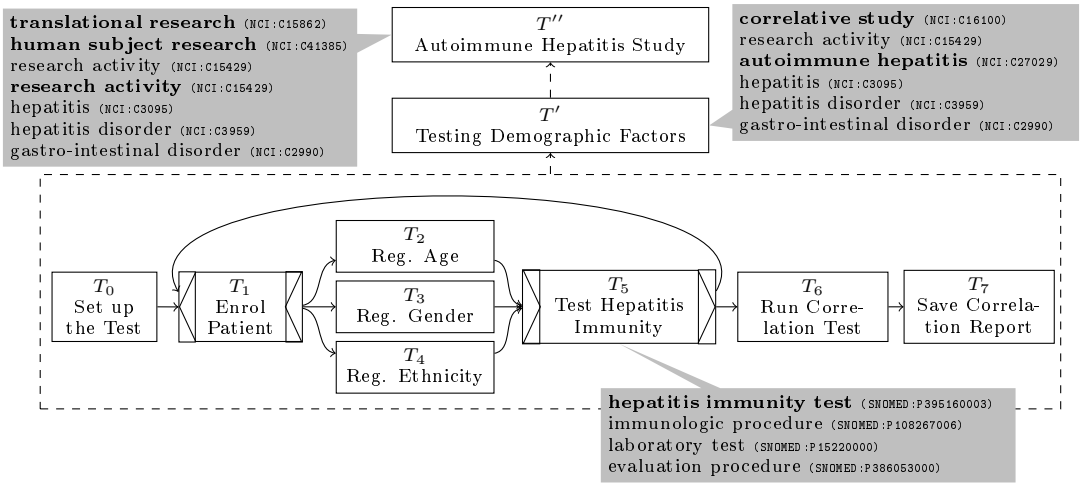

Figure 4.2: The definition of the HN and the labelling function for workflow W_1. The boldfaced labels are assigned by the workflow designers and the normal-font labels result from the ontological relations in the vocabulary.

Workflow Policy There are many constraints about the workflow. For simplicity of the demonstration, we only consider the following policy for the sake of simplicity. This constraint states that the user assigned to task T_1 (represented with the constant symbol $\mathtt{T}_1$) must have the role *senior researcher*.

$$\phi_5 \equiv \forall^{TASKI} t \ ((\mathtt{Task\text{-}Type\text{-}Of}(t) = \mathtt{T}_1) \rightarrow \mathtt{Sen\text{-}Res}(\mathtt{User\text{-}Of}(t))) \tag{4.2.5}$$

Event-Flow In this section we present the event flow for policy enforcement in the course of workflow definition, workflow instantiation and an sample task instantiation. The initial system state, s_0, is as the following:

$$\Phi = \{\phi_1, \phi_2, \phi_3\}$$
$$\mathcal{A} = \{\cdots\} \qquad \textit{(global assertions e.g. user roles.)}$$

We assume $\Gamma(s_0)$ is satisfiable, so, the system starts in a compliant state.

Workflow Definition First assume a user Alice who is not qualified for the role *Admin* sends the request for defining W_1. We use the constant $\mathtt{W}_1$ to represent W_1. The assertions of $r(s)$ will include:

$$A \equiv \neg\mathtt{Admin}(\mathtt{Alice}), \ \ \mathtt{Definer\text{-}Of}(\mathtt{W}_1) = \mathtt{Alice}$$

which implies

$$\exists^W w\ \neg\texttt{Admin}(\texttt{Definer-Of}(w))$$

Consider ϕ_1:

$$\phi_1 \equiv \forall^W w\ \ \texttt{Admin}(\texttt{Definer-Of}(w))$$

Thus, A implies $\neg\phi$, so $A \wedge \phi_1 \equiv \bot$, and therefore, $\Gamma(r(s)) \equiv \bot$, i.e is not satisfiable. So, $r(s)$ is not a compliant state, and thus, the request is rejected.

Now, assume another user Bob who is qualified for the role *Admin* sends the request for defining W_1 with the corresponding policy ϕ_4. The assertions and policies of $r(s) = \langle \mathcal{A}', \Phi' \rangle$ will be as the following. We use the constants $\texttt{T}'$, $\texttt{T}''$, and $\texttt{T}_i$'s for the corresponding tasks. We have only shown the assertions regarding the purpose vocabulary terms *research* and *immunologic procedure*, using the predicate $\texttt{Attr}_\texttt{r}$ and $\texttt{Attr}_\texttt{ip}$ respectively. Other vocabulary terms are represented similarly.

$$\begin{aligned}
\Phi' =& \Phi \cup \{\phi_5\}\\
\mathcal{A}' =& \mathcal{A} \cup \{\texttt{Definer-Of}(\texttt{W}_1) = \texttt{Bob}\\
& \texttt{WF-Of}(\texttt{T}'') = \texttt{W}_1, \texttt{WF-Of}(\texttt{T}') = \texttt{W}_1, \texttt{WF-Of}(\texttt{T}_1) = \texttt{W}_1, \cdots, \texttt{WF-Of}(\texttt{T}_7) = \texttt{W}_1,\\
& \texttt{Attr}_\texttt{r}(\texttt{T}''), \texttt{Attr}_\texttt{r}(\texttt{T}'), \neg\texttt{Attr}_\texttt{r}(\texttt{T}_1), \cdots, \neg\texttt{Attr}_\texttt{r}(\texttt{T}_7),\\
& \neg\texttt{Attr}_\texttt{ip}(\texttt{T}''), \neg\texttt{Attr}_\texttt{ip}(\texttt{T}'), \neg\texttt{Attr}_\texttt{ip}(\texttt{T}_1), \cdots, \texttt{Attr}_\texttt{ip}(\texttt{T}_5), \cdots, \neg\texttt{Attr}_\texttt{ip}(\texttt{T}_7), \cdots\}
\end{aligned}$$

Since $\Gamma(r(s))$ is satisfiable, the request is accepted and the system state is updated.

Workflow Instantiation Assume Alice sends the request for instantiating W_1. The assertions and policies of $r(s) = \langle \mathcal{A}', \Phi' \rangle$ will be the following. Note that for simplicity we assume the workflow has no parameter other than its instantiator.

$$\begin{aligned}
\Phi' =& \Phi\\
\mathcal{A}' =& \mathcal{A} \cup \{\texttt{WF-Type-Of}(\texttt{N}_1) = \texttt{W}_1, \texttt{WF-Admin-Of}(\texttt{N}_1) = \texttt{Alice}, \neg\texttt{Admin}(\texttt{Alice})\}
\end{aligned}$$

Since $\Gamma(r(t))$ is satisfiable, the request is accepted and the system state is updated.

Task Instantiation Assume Alice sends the request for instantiating task T_1. Also, assume that besides the task instantiator, there are two other parameters for the task denoting the assigned user and data item, whose values are respectively Alice and D_1. We use the constant $\texttt{D}_1$ to represent data item D_1.

The assertions and policies of $r(s) = \langle \mathcal{A}', \Phi' \rangle$ will be as the following. We use the fresh constant $\texttt{K}_1$ to represent the task instance.

$$\begin{aligned}
\Phi' =& \Phi \cup \{\phi_4\}\\
\mathcal{A}' =& \mathcal{A} \cup \{\texttt{Task-Type-Of}(\texttt{K}_1) = \texttt{T}_1, \texttt{WFI-Of}(\texttt{K}_1) = \texttt{N}_1\\
& \texttt{Task-Admin-Of}(\texttt{K}_1) = \texttt{Alice}, \texttt{User-Of}(\texttt{K}_1) = \texttt{Alice}, \texttt{Data-Of}(\texttt{K}_1) = \texttt{D}_1\}
\end{aligned}$$

Note that the assertions for Alice's roles are already in $\mathcal{A}$ and we assume there are no attributes for D_1.

Consider the following steps to evaluate $\texttt{Purpose}_{\psi_2}(\texttt{T}_1)$, according Section 4.2:

- Considering the assertion $\texttt{WF-Of}(\texttt{T}_1) = \texttt{W}_1$ which is already in A, the HN corresponding to W_1 if extracted.
- Considering the assertions $\texttt{Attr}_\texttt{a}(\texttt{T}_i)$ and $\texttt{Attr}_\texttt{ip}(\texttt{T}_i)$ corresponding to the T_i's that belong to W_1, the atomic propositions r and ip are considered and the *labelling function* is constructed as:

$$L = \{(T'', r), (T', r), (T_5, ip)\}$$

- The purpose model-checking algorithm is run with the hierarchy net of W_1 (let us call it $\mathcal{H}_1$) and the labelling function as constructed above (let us call it L_1). According to the result, $T_1 \notin [\![\psi_2]\!]_{\mathcal{H}_1,L_1}$, therefore, $\texttt{Purpose}_{\psi_2}(\texttt{T}_1) \equiv \bot$.

Now, consider the following assertions:

$$A \equiv \texttt{Data-Of}(\texttt{K}_1) = \texttt{D}_1 \wedge \texttt{Task-Type-Of}(\texttt{K}_1) = \texttt{T}_1$$

Recalling ϕ_4:

$$\forall^{TASKI} t \ \big((\texttt{Data-Of}(t) = \texttt{D}_1) \rightarrow \texttt{Purpose}_{\psi_2}(\texttt{Task-Type-Of}(t))\big)$$

we can see that $A \wedge \phi_4$ implies $\texttt{Purpose}_{\psi_2}(\texttt{T}_1)$ which according to the evaluation above is equivalent to $\bot$. Therefore $A \wedge \phi_4 \equiv \bot$, and hence, $\Gamma(r(t)) \equiv \bot$, i.e. $r(t)$ is not a compliant state and the request is rejected.

4.3 Purpose-Based Obligations

Obligations are commitments that result from granting an access [9]. A purpose-based obligation is an obligation that is triggered based on some purpose constraints. For example, an obligation may require the record to be *anonymized* before access for *research* purposes is allowed. Or, a fee might be charged, or notification/audit information produced, every time some personal data is used for the purpose of *marketing*. We have not included obligations in the ACPL, but once purpose constraints are supported in an access control languages which supports obligations, such as XACML, purpose-based obligations can be modelled straightforwardly.

5

Implementing a WfRM

It is a well-known result that evaluating the satisfiability of firs-order logic formulas in its general form is undecidable [10]. But with making some restrictive assumptions we can evaluates certain types of assertions against policies efficiently by model checking. This makes it possible to check whether a given request violates any of the policies of the system, on which basis the request can be accepted or rejected. This is a very practical result for handling access control requests in the system although it does not solve the problem of *policy satisfiability analysis*, i.e. checking whether the policies contradict each other and whether they can allow any requests at all.

In this chapter, we show how a simple WfRM, with restrictions on the form of the policy and assertions, can be implemented using XACML. The eXtensible Access Control Markup Language ("XACML") is a very common access control product in the industry and has been used by many organization over the past decade [17]. It provides an XML-based open standard for encoding access control policies and requests/responses, as well as (informal) operational semantics for verifying requests against the policies and making access control decisions.

We discuss the restrictive assumptions in Section 5.1. Then, in Section 5.2 we discuss the XACML's policy form and processing model as a special case of the general model presented in Chapter 4. In Section 5.3 we discuss how XACML can be used as a basis for implementing a simple form of WfRM which is subject to our restrictive assumptions. Finally we revisit the examples of Section 4.2 and show the event-flow of policy analysis for these examples in the XACML implementation.

5.1 Restrictions on the Form of Policies and Assertions

In order to make it possible to implement the WfRM using XACML, we make some restrictive assumptions about the form of assertions and policies. We will show in our examples that these restricted forms are sufficient for covering most types of policies.

Restrictions on the Form of Assertions We assume that the assertions are only *attribute assertions.*

Definition 5. *An* attribute assertion *is a ground formula in one of the following forms:*

- $C_1 = F(C_2)$ *in which* F *is a function and* C_1 *and* C_2 *are constants of suitable types. This basically says the value of attribute* F *for* C_2 *is* C_1*, or in other words,* C_1 *has a relation of type* F *with* C_2*.*
- $P(C_1)$ *in which* P *is a predicate and* C_1 *is a constant of the suitable type. This basically says the value of the boolean attribute* P *for* C_1 *is true.*
- $\neg P(C_1)$ *in which* P *is a predicate and* C_1 *is a constant of the suitable type. This basically says the value of the boolean attribute* P *for* C_1 *is false.*

Restrictions on the Form of Policies We assume the following restrictions on the form of policies:

1. Every policy formula contains only one universal quantifier, i.e. has the form $\forall^y x\phi$.
2. The variable type in the policy is either *workflow*, *workflow instance*, *task*, or *task instance*, i.e. $y \in \{WF, WFI, TASK, TASKI\}$.

These restrictions ensure that the variable ranges over relatively small sets since the number of users and data items may be very large, but workflow-related entities are far less many. Also, it makes sense to assume that the policies of a *workflow* system make only universal rules about the workflow-related entities and other entities such as *users* and *data* are relevant only as long as they are related to the workflow-related entities.

By restricting the number of variables to one, we basically restrict policies to individual *workflows*, (respectively, *workflows instances*, *tasks*, and *task instances*) and rule out policies that span over different *workflows* (respectively, *workflow instances*, *tasks*, and *task instances*). For example, it is not possible to make policies such as:

- "The instantiator of a task cannot instantiate any other tasks or workflows while the task is not completed."
- "Different users must be assigned as actor to different instances of each task."

Many common access control policies can be expressed under the above restrictions. These restrictions are important in making it possible to test whether assertions about one single workflow-related entity (e.g. a task) refute a given policy, and provide a simple mechanism to test the satisfiability of policies at the time when a new workflow-related entity is created, by checking if any of the assertions about its attributes refutes any of the policies.

5.2 XACML Architecture

In this section we discuss the operational model of XACML and match it with the abstract model we defined above. This provides an introduction to discuss the details of how to implement a WfRM based on XACML in Chapter 5.

5.2.1 XACML Request

An *XACML request* [1] is a ground formula, formed from the conjunction of a number of *attribute assertions*.

5.2.2 XACML Policy

XACML policies have the following form [2]:

$$\phi \equiv \forall^{y_1} x_1 \cdots \forall^{y_n} x_n \ \phi'$$
$$\phi' \equiv \textbf{Target}(x_1, \cdots, x_n) \wedge \textbf{Condition}(x_1, \cdots, x_n) \qquad \text{(where } x_1, \cdots, x_n \text{ are free variables in } \phi'.\text{)}$$

Additionally, every XACML policy is assigned a type *Deny* or *Permit* which specifies the type of result that must be produced if the policy is evaluated to true. Usually, XACML policies contain at most four variables of types *subject*, *resource*, *action* and *environment*, but XACML version 3.0, the most recent version of the standard, has relaxed this limitation and there can be any number of variable types (called *categories* in XACML terms) [17].

[1] Not to be confused with systems requests defined in Sectoin 4.1.

[2] The **Target** subformula can only take simple forms while the **Condition** can be more complex. The reason for this distinction is to avoid performing the more complex computations of the **Condition**, should the simpler **Target** evaluate to false. In other words, as the name implies **Target** provides a simple applicability criteria for the policy, and is used as a filtering mechanism to test only the policies that are *applicable* to a XACML request.

5.2.3 XACML Policy Evaluation

Upon receiving a XACML request, the XACML engine *attempts* to build a *model* $\mathcal{M}$ based on the assertions in the XACML request (See Appendix A for the definition of a model for ACPL). We discuss the details of building the model later below.

Using this model, it is tested whether $\mathcal{M} \models \phi$ holds true (See Appendix A for the definition of the semantics for ACPL); a result is produced according to the following procedure:

- If the information in $\mathcal{M}$ is not sufficient to evaluate $\mathcal{M} \models \phi$ the result is *Indeterminate* [3]. This situation results from missing information in the model, for example, lack of a value for a function, and happens if after replacing the values for the variables, predicates and functions, the resulting formula still contains undefined predicates or functions whose values are not determined.
- If $\mathcal{M} \not\models \phi$ the result is *Not Applicable*.
- If $\mathcal{M} \models \phi$, the result is *Deny* or *Permit* depending on the type of policy.

The important case, is the case of $\mathcal{M} \not\models \phi$. If we show the assertion formulas in the XACML request as A, we have $\mathcal{M} \models A$ and $\mathcal{M} \models \neg\phi$ in this case.

Now notice that since A is a ground formula, any other model that satisfies A includes the same set of concrete objects (possibly with different names) and similar interpretations that refutes ϕ. Thus, for any model $\mathcal{M}$, if $\mathcal{M} \models A$, then we have $\mathcal{M} \models \neg\phi$. This, according to the definition of the semantics implies, $\mathcal{M} \models (A \rightarrow \neg\phi)$, which leads to $\mathcal{M} \models \neg(A \wedge \phi)$. In other words, $\neg(A \wedge \phi)$ is satisfied by every possible model and is a *valid formula*, thus, $A \wedge \phi$ is not *satisfiable*.

This result provides a building block for using XACML to check whether the system state is *compliant* as we will discuss further in Section 5.3.2.

Building the Model The process for making the model is as follows:

- Each constant is mapped to a distinct concrete object of the same type.
- The set of entities for each type is set to include all the concrete objects of that type.
- For each assertion of the form $\mathtt{C}_1 = \mathtt{F}(\mathtt{C}_2)$ in which $\mathtt{C}_1$ and $\mathtt{C}_2$ are constants and $\mathtt{F}$ is a function, the pair (e_1, e_2) is added to the interpretation of $\mathtt{F}$ in which e_1 and e_2 are the corresponding concrete objects to which $\mathtt{C}_1$ and $\mathtt{C}_2$ are mapped.
- For each assertion of the form $\mathtt{P}(\mathtt{C}_1)$ in which $\mathtt{C}_1$ is a constant and $\mathtt{P}$ is a predicate, (e_1, true) is added to the interpretation of $\mathtt{P}$ in which e_1 the concrete object to $\mathtt{C}_1$.
- For each assertion of the form $\neg\mathtt{P}(\mathtt{C}_1)$ in which $\mathtt{C}_1$ is a constant and $\mathtt{P}$ is a predicate, (e_1, false) is added to the interpretation of $\mathtt{P}$ in which e_1 the concrete object to $\mathtt{C}_1$.

For example, when an assertion such as $\mathtt{Admin}(\mathtt{Alice})$ is received the following steps take place to form the model $\mathcal{M}$:

- The set $\mathit{USER}^{\mathcal{T}}$ is formed as the set of concrete objects for the type *USER*.
- The interpretation of the constant $\mathtt{Alice}$ is set to the concrete object *Alice*.
- The concrete object *Alice* is added to the set $\mathit{USER}^{\mathcal{T}}$.
- The set $\mathtt{Admin}^{\mathcal{M}}$ is formed as the interpretation of the $\mathtt{Admin}$ predicate.
- The $(\mathit{Alice}, \mathit{true})$ is added to $\mathtt{Admin}^{\mathcal{M}}$.

[3] XACML considers different types *Indeterminate*; for the sake of brevity we consider only one type of *Indeterminate*.

	Deny	**Permit**	**Indeterminate**	**Not Applicable**
Not Applicable	Deny	Permit	Indeterminate	Not Applicable
Indeterminate	Deny	Indeterminate	Indeterminate	
Permit	Deny	Permit		
Deny	Deny			

Figure 5.1: Combining results from evaluationg different policies using *Deny Overrides* for two results. The case for more than two results is handled by pairwise combining.

Built-in Functions and Predicates The truth value of some standard predicates and functions are computed by the XACML engine and added to any model when necessary. For example, comparison predicates (such as $\geq$) are supported by XACML and their truth values can be computed for any model. In other words, the interpretation of these predicates and functions is added by the XACML to any model that contain them and need not be made based on the given assertions in the XACML request.

By adding $\texttt{Purpose}_\psi$ as standard predicates to the XACML engine, the truth value for $\texttt{Purpose}_\psi$ family of predicates can be computed by XACML similar to other built in predicates. For encoding and evaluating purpose constraints, we use the custom XACML function `evaluate-modal-formula` which takes a policy formulated as a PML formula and a task ID, and returns a boolean value resulting from running the purpose model-checking algorithm. This basically gives the evaluation for the predicate $\texttt{Purpose}_\psi(\texttt{T})$ for some task represented as `T`. The modal logic formula which represents a policy is encoded as a custom XACML type `modal-formula` whose value is a straightforward XML encoding of the formula. See Figure 5.2 for an example.

This XACML formula is evaluated in a similar way to what we discussed in Section 4.2 with some difference for improving the efficiency and for simplifying XACML requests and policy forms. First, we assume each task has a unique identifier and the system keeps a mapping of tasks to workflow definition IDs so that this need not be parsed from the `WF-Of` assertions. Moreover, we assume the attributes of tasks, i.e. the *labelling function* is stored in the system together with the workflow definition and need not be parsed from the requests. For evaluating a PML formula, the HN and the labelling function are retrieved based on the task ID and the purpose-model checker is executed and the result, which is a true or false value is returned to XACML engine.

5.2.4 XACML Response

Similar to individual policy evaluations, the eventual response to a request in XACML can be either *Permit*, *Deny*, *Indeterminate* or *Not Applicable*. XACML has a mechanism for deciding how the results from evaluating different policies are combined into a *response* when policies are collected within a *policy set*. The most common types of combining mechanism for *policy sets* are *Permit Overrides* and *Deny Overrides* which give priority to, respectively a *Permit* or *Deny* result. Figure 5.1 shows how combining works for the case of *Deny Override*.

5.3 Implementing WfRM based on XACML

The XACML engine cannot perform satisfiability analysis in the general case. Neither is it capable of performing the consistency checks required for the *integrity condition*, i.e. whether policies are consistent. In fact, testing satisfiability of the ACPL formulas is known to be undecidable in the general case. However, subject to the limitations we assumed for the policies and assertion, as we showed in the previous section, it can check whether a given assertion violates a given policy. On this basis, we propose a model for implementing a limited form of WfRM based on XACML.

5.3.1 XACML Request

In order to encode the assertions, we use the *Resource Description Framework* ("RDF") format [13] which is syntactically richer than the standard XACML request format. The XACML request format allows encapsulating the entire RDF code within any of its elements. Figure 5.6 shows a sample RDF encoding of some assertions.

5.3.2 XACML Policy Structure

For every policy ϕ_i of the form $\forall^y x \phi_i'$ in the system [4], we create a corresponding XACML policy of type *Deny* that encodes $\forall^y x \neg\phi_i'$. As we discussed in Section 5.2.3, verifying this policy against a XACML request carrying the assertion A leads to a *Deny* result if and only if $A \wedge \phi_i$ is not *satisfiable*. So, we have a mechanism to test the satisfiability of $A \wedge \phi_i$.

Assuming the set of all policies is $\Phi = \{\phi_1, \cdots, \phi_n\}$, the WfRM needs to test whether the formula $\gamma = A \wedge \phi_1 \wedge \cdots \wedge \phi_n$ is satisfiable in order to decide whether to allow or reject an access. This formula can be rearranged as $\gamma = (A \wedge \phi_1) \wedge \cdots \wedge (A \wedge \phi_n)$ and thereby:

$$\neg\gamma = \neg(A \wedge \phi_1) \vee \cdots \vee \neg(A \wedge \phi_n) \tag{5.3.1}$$

Obviously, if any of $\neg(A \wedge \phi_i)$ be a valid formula, $\neg\gamma$ is also valid; in other words, checking the satisfiability of γ can be done by checking the satisfiability of the assertions with each policy.

Now, consider the semantics of *Deny Overrides* given in Figure 5.1. If we collect all the policies in a *policy set* with the *Deny Overrides* combining algorithm, an overall *Deny* response will be produced if checking A against any of the ϕ_i's returns a *Deny*, i.e. if $A \wedge \phi_i$ is unsatisfiable. Note that *Deny Override* in fact implements the result of the disjunctions in Formula 5.3.1 above.

Thus, if we encode every policy of the form $\forall^y x \phi_i'$ as an XACML policy of *Deny* type which encodes $\forall^y x \neg\phi_i'$, and then collect all the policies of the system within a *policy set* with the combining mechanism *Deny Override* it will give us a *Deny* response if γ is unsatisfiable, i.e. it produces a *Deny* response whenever a request must be denied.

Figures 5.4 shows an example of a policy encoded in XACML corresponding to the consent directive of the patient discussed in Section 4.2.

Extracting Assertions XACML has standard mechanisms for parsing the values of the attributes from the XACML request. Since we have used RDF for encoding assertions, we cannot use them for reading the assertions and we need an XML query mechanism. XACML supports XPath [19] queries natively, and since we are using RDF, an RDF query language such as SPARQL [15] is also a suitable mechanism which can be used in the form of custom functions implemented as an extension to XACML core. We do not discuss the details of this any further and assume that a suitable XML querying mechanism for parsing the assertions is used.

For better readability, we assume such queries are each encapsulated within a XACML *policy variable*[5]. XACML policy variables are macro-like mechanisms for encapsulating a predicate or function in a named variable definition so that it can be referenced elsewhere in the XACML policy. Some examples of defining and referencing XACML variables are shown in Figures 5.2, 5.3, and 5.4.

5.3.3 Policy Evaluation

The overall procedure is that in the event of each request for defining the workflow, instantiation of a workflow, or instantiation of a task, the assertions corresponding to the new entity being created (i.e. the workflow definition, workflow instance, or task instance) are sent to the XACML engine in the form of an XACML request. As discussed above, in case of a `Deny` response, the assertions are incompatible with the

[4] Note that as mentioned in Section 5.1, we assume that policies contain only one universal quantifier and one variable.

[5] Not to be confused with *variables* in ACPL.

```
<VariableDefinition VariableId="Purpose-Predicate-1">
  <Apply FunctionId="evaluate-modal-formula">
   <AttributeValue DataType="modal-formula">
    <Not>
     <Or>
      <ADiamond>
       SNOMED:P108267006
      </ADiamond>
      <FDiamondPossible>
       SNOMED:P108267006
      </FDiamondPossible>
     </Or>
    </Not>
   </AttributeValue>
   <VariableReference VariableId="Task-Type-Of"/>
  </Apply>
</VariableDefinition>
```

the formula:
$\neg(\langle \mathbf{A} \rangle ip \ \vee \ \langle \mathbb{F} \rangle ip)$
(must not be possibly for *immunologic procedure*)

Reference to the task definition.

Figure 5.2: XACML encoding of the purpose constraint corresponding to ψ_1 from Formula 4.2.3.

```
<Rule Effect="Deny">
 <Target/>
 <Condition FunctionId="function:not">
  <Apply FunctionId="function:or">
   <Apply FunctionId="function:not">
    <VariableReference VariableId="Purpose-Predicate-1"/>
   </Apply>
   <VariableReference VariableId="Sen-Res-User-Of"/>
  </Apply>
 </Condition>
</Rule>
```

Deny if the following policy does not hold:

$\texttt{Purpose}_{\psi_1}(\texttt{Task-Type-Of}(t)) \rightarrow \texttt{Sen-Res}(\texttt{User-Of}(t))$

Figure 5.3: XACML encoding of the overarching policy of Formula 4.2.3. The purpose predicate is defined as an XACML variable in Figure 5.2.

```
<Rule Effect="Deny">
 <Target>
  <AnyOf>
   <AllOf>
    <Match MatchId="function:string-equal">
     <AttributeValue DataType="string">
      D1
     </AttributeValue>
     <VariableReference VariableId="Data-Of"/>
    </Match>
   </AllOf>
  </AnyOf>
 </Target>
 <Condition FunctionId="function:not">
  <VariableReference VariableId="Purpose-Predicate-2"/>
 </Condition>
</Rule>
```

Deny if the following policy does not hold:

$\texttt{Data-Of}(t) \ = \ \texttt{D}_1$, and

$\neg\texttt{Purpose}_{\psi_2}(t)$

Figure 5.4: XACML encoding of the consent directive from Formula 4.2.4 (equivalently $\forall^{TASKI} \neg \texttt{Data-Of}(t) \ = \ \texttt{D}_1 \vee \texttt{Purpose}_{\psi_2}(t)$). The purpose constraint ψ_2 is encoded as an XACML variable in a similar way to Figure 5.2.

policy and the request is rejected. Other responses, i.e. *Not Applicable* or *Indeterminate*, indicate that the assertions are not incompatible with the policy and the request is accepted [6].

Note that since we made the assumptions that there is only one variable in each policy, and thus, there are no policies that are about more than one entity, it is sufficient to include all the assertions that are

[6] Note that this makes the software development a bit difficult since XACML produces an *Indeterminate* result both in the case of a missing value (i.e. insufficient information) and an error. We assume that other mechanisms are used to distinguish these cases.

related to the entity being created and the rest of the assertions in the system need not be sent over the XACML request, because they will not affect checking the satisfiability.

5.4 Event-Flow

In this section we present the event flow for policy enforcement in a WfRM implemented based on XACML for the same example as given in Section 4.2. We only discuss the case of workflow definition and task instantiation as the other cases are quite similar.

Workflow Definition Similar to Section 4.2, first assume a Alice, not qualified for the admin role, sends the request for defining W_1. The resulting assertions are as the following; the request corresponding to these assertions is shown in Figure 5.6:

$$\neg\texttt{Admin}(\texttt{Alice}), \texttt{Definer-Of}(\texttt{W}_1) = (\texttt{Alice})$$

The workflow definition request also includes a workflow-specific policy Formula 4.2.5, encoded in XACML in Figure 5.5:

$$\phi_5 \equiv \forall^{TASKI} t \ \ ((\texttt{Task-Type-Of}(t) = \texttt{T}_1) \rightarrow \texttt{Sen-Res}(\texttt{User-Of}(t)))$$

Also, consider the overarching policy of Formula 4.2.1 which is encoded in XACML in Figure 5.7:

$$\phi_1 \equiv \forall^{W} w \ \ \texttt{Admin}(\texttt{Definer-Of}(w))$$

Upon receiving this request, the XACML engine evaluates it against all the policies, including the ones in Figure 5.5 and 5.7. The evaluation for the former leads to an *Indeterminate* result since neither the XACML variable references in the *Match* nor the one in the *Condition* element can be resolved against the given request. We disregard the other overarching policies for simplicity but their evaluation similarly leads to the same *Indeterminate* result.

For the policy of Figure 5.7, the XACML variable reference within the *Condition* element is parsed by running the corresponding XML queries, leading to the extraction of the value *false*. So the condition evaluates to *true* and the policy produces a *Deny* response. Since the high-level combining procedure is *Deny-Overrides*, the eventual response will definitely be *Deny* and the request will be rejected.

Now, consider the alternative case where *Bob* makes the same request. In this case, since the *Condition* evaluates to false, the result from evaluating this policy will be *Not Applicable* and assuming there is no *Deny* from other policies in the system, the request will be accepted.

Note that there is another aspect for checking the workflow definition request that cannot be handled by XACML: evaluating whether the workflow-specific policy (from Formula 4.2.5) is consistent with the rest of the policies in the system (i.e. their conjunction is satisfiable). This cannot be handled by XACML and should be taken care of by other means.

Task Instantiation Similar to Section 4.2, assume Alice sends the request for instantiating task T_1; the assigned user and data to the task instance are respectively *Alice* and D_1.

At this point, the policies in the system are the ones from Formulas 4.2.1, 4.2.3, 4.2.5, respectively encoded in XACML in Figures 5.7, 5.3, and 5.5. Moreover, since D_1 is being assigned to the task instance, its *Consent* policy from Formula 4.2.4, encoded in Figure 5.4, is also considered. In the interest of simplicity, we disregard the other overarching policy from Formula 4.2.2.

The XACML request pertaining to this task instantiation request which includes all the assertions related to the instance is shown in Figure 5.8. Evaluating this XACML request against each of the policies of Figures 5.7, 5.3, and 5.5, leads to a *Indeterminate* result. Evaluating against the policy of Figure 5.4, however, leads to a *Deny*, since the XACML variable reference *purpose predicate-2* returns false after running the purpose model-checking algorithm on ψ_2 and T_1, the task type of the task instance being created. Thus, the request is rejected.

```
<Rule Effect="Deny">                                   → Deny if the following policy does not hold:
 <Target>
  <AnyOf>
   <AllOf>
    <Match MatchId="function:string-equal">
     <AttributeValue DataType="string">
      T1                                               → Task-Type-Of(t) = T1, and
     </AttributeValue>
     <VariableReference VariableId="Task-Type-Of"/>
    </Match>
   </AllOf>
  </AnyOf>
 </Target>
 <Condition FunctionId="function:not">
  <VariableReference VariableId="Sen-Res-User-Of"/>    → ¬Sen-Res(User-Of(t))
 </Condition>
</Rule>
```

Figure 5.5: XACML encoding of the workflow-specific policy from Formula 4.2.5 (equivalently $\forall^{TASKI} t\ \neg\texttt{Task-Type-Of}(t) = \texttt{T}_1 \vee \texttt{Sen-Res}(\texttt{User-Of}(t))$).

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:acpl="http://ucalgary.ca/cpsc/jafarim/thesis/acpl">
 <acpl:USER about="Alice">
  <acpl:Admin>false</acpl:Admin>                       → ¬Admin(Alice)
 </acpl:USER>
 <acpl:WF about="W1">
  <acpl:Definer-Of rdf:resource="Alice"/>              → Definer-Of(W1)=Alice
 </acpl:WF>
</rdf:RDF>
```

Figure 5.6: The request assertions for defining W_1 by *Alice*.

```
<Rule Effect="Deny">                                   → Deny if the following policy does not hold:
 <Target/>
 <Condition FunctionId="function:not">
  <VariableReference VariableId="Admin-Definer-Of"/>   → ∀^W w Admin(Definer-Of(w))
 </Condition>
</Rule>
```

Figure 5.7: XACML encoding of the policy from the Formula 4.2.1.

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:acpl="http://ucalgary.ca/cpsc/jafarim/thesis/acpl">
 <acpl:USER about="Alice">
  <acpl:Admin>false</acpl:Admin>                       → ¬Admin(Alice), Sen-Res(Alice)
  <acpl:Sen-Res>true</acpl:Sen-Res>
 </acpl:USER>
 <acpl:TASKI about="K1">
  <acpl:Task-Type-Of rdf:resource="T1"/>               → Task-Type-Of(K1)=T1
  <acpl:WFI-Of rdf:resource="N1"/>                       WFI-Of(K1)=N1
  <acpl:Task-Admin-Of rdf:resource="Alice"/>             Task-Admin-Of(K1)=Alice
  <acpl:User-Of rdf:resource="Alice"/>                   User-Of(K1)=Alice
  <acpl:Data-Of rdf:resource="D1"/>                      Data-Of(K1)=D1
 </acpl:TASKI>
</rdf:RDF>
```

Figure 5.8: The request assertions for instantiating T_1 by *Alice* and assigning *Alice* and D_1 to it.

6

Parametrized Purposes

Consider the simple case that the policy allows using patient's record for *treatment* purposes. What is naturally meant by this policy is that the purpose must be the treatment of *the same patient* whose record is being used. If Alice's record, for example, is used by a physician to help with Bob's treatment (say for some genetic analysis if they are siblings), this does not match the intuitive intention of the policy although access is still for the purpose of *treatment*.

This example shows that in reality, there is not one single *treatment* purpose; rather, there is a family of different purposes of the form "*treatment-of*(x)" where x represents a patient. In other words, the simple purpose of treatment can be parametrized to differentiate between treatments of different patients. Alice may wish to restrict access to her record only to her own treatment, or perhaps to her children and close family members. There are various other examples for parametrized purposes in different domains; for example, a customer can specify that her/his home address can only be used for *delivery* of the ordered product as opposed to delivery of other things, such as marketing material.

In order to support such policies, we need to model each purpose as a predicate and introduce variables into the language as the following, in which x represent any variable.

$$\psi ::= \top \mid P(x, \cdots, x) \mid (\neg\psi) \mid (\phi \wedge \psi') \mid (\langle \mathbf{A} \rangle \psi) \mid (\langle \mathbf{F} \rangle \psi) \mid (\langle \mathbb{F} \rangle \psi)$$

The semantics of this language is defined based on a model $\mathcal{N}$ which is defined as follows:

- A non-empty set $\mathcal{O}$ which represents a *universe* of concrete objects.
- A variable lookup table which assigns a concrete object o to each variable.
- An *interpretation* which defines the predicates at each transition and assigns concrete objects to the variables. For each n-ary predicate P, its interpretation is shown as $P^{\mathcal{N}} \subseteq \mathcal{O}^n \times \mathcal{T}_{\mathcal{H}} \times \{true, false\}$ in which $\mathcal{T}_{\mathcal{H}}$ is the set of all transitions in $\mathcal{H}$. The interpretation of each predicate assigns a truth value to each permutation of concrete values (with suitable artiy) at each transition. This is essentially an extension of the *labelling function* in our language.

On its basis, the semantics of the language is defined as $[\![\psi]\!]_{\mathcal{N}}$ which represents the set of transitions satisfying ψ in the context of the model $\mathcal{N}$. The semantics definition for this language basically replaces the variables with their values and then evaluates each purpose predicate at each task. Once the variables are replaced, each predicate becomes equivalent to an atomic proposition and thus the original definition of the

semantics of the purpose language can be applied. This is formalized below:

$$\begin{aligned}
[\![\top]\!]_{\mathcal{N}} &= \mathcal{T}_{\mathcal{H}}.\\
[\![\neg\psi]\!]_{\mathcal{N}} &= \mathcal{T}_{\mathcal{H}} \backslash [\![\psi]\!]_{\mathcal{N}}.\\
[\![\psi_1 \wedge \psi_2]\!]_{\mathcal{N}} &= [\![\psi_1]\!]_{\mathcal{N}} \cap [\![\psi_2]\!]_{\mathcal{N}}.\\
[\![P(o_1, \cdots, o_n)]\!]_{\mathcal{N}} &= \{t \in \mathcal{T}_{\mathcal{H}} \mid \langle o_1, \cdots, o_n, t, \mathit{true}\rangle \in P^{\mathcal{N}}\}.\\
[\![\langle \mathbf{A}\rangle\psi]\!]_{\mathcal{N}} &= [\![\psi]\!]_{\mathcal{N}} \cup PRE_A([\![\langle \mathbf{A}\rangle\psi]\!]_{\mathcal{N}}).\\
[\![\langle \mathbf{F}\rangle\psi]\!]_{\mathcal{N}} &= [\![\psi]\!]_{\mathcal{N}} \cup PRE_F([\![\langle \mathbf{F}\rangle\psi]\!]_{\mathcal{N}}.\\
[\![\langle \mathbb{F}\rangle\psi]\!]_{\mathcal{N}} &= [\![\psi]\!]_{\mathcal{N}} \cup PRE_F^{\exists}([\![\langle \mathbb{F}\rangle\psi]\!]_{\mathcal{N}}).
\end{aligned}$$

We assume $\vee$ and $\rightarrow$ are defined as derived forms based on $\neg$ and $\wedge$. Note that we do not define quantifiers for this language and assume that quantifiers from the upper-level access control language can be used to range the variables as desired as will be shown in the example below.

We show the formulas in this language as $\psi(x_1, \cdots, x_n)$ to denote the variables in ψ.

As an example, consider the policy that stipulates a patient's record can only be used for her or his own *treatment*. Assuming patients are represented by the type PAT and `Owner-Of` is a function that returns the owner of a health records, this policy can be formulated as:

$$\forall^{TASKI} t\ \forall^{PAT} p\ \forall^{PAT} q\ \Big(\Big(\texttt{Owner-Of}(\texttt{Data-Of}(t)) = p \wedge \texttt{Purpose}_{\psi_3(x)}(\texttt{Task-Type-Of}(t), q)\Big) \rightarrow p = q\Big)$$

in which $\psi_3(x) \equiv \langle \mathbf{A}\rangle \mathrm{tr}(x) \vee \langle \mathbb{F}\rangle \mathrm{tr}(x)$.

The predicate for modelling this attribute, $\texttt{Attr}_{\texttt{tr}}$ has the signature $(TASK, PAT)$, so, if task T is assigned the data of Alice, the following attributes assertion is added to the system state: $\texttt{Attr}_{\texttt{tr}}(\texttt{T}, \texttt{Alice})$.

Note that the predicates $\texttt{Purpose}_{\psi}$ and $\texttt{Attr}_{v}$ do the mapping from the workflow parameters to the underlying purpose-parameters. Especially, if the purpose formula has more than one variable, the order of the assignment is important. The choice of mapping these parameters and the consistency between the mapping performed by each of $\texttt{Purpose}_{\psi}$ and $\texttt{Attr}_{v}$ is a crucial modelling decision and is important in capturing the intention of the policies and producing the desired outcomes. To keep this simple, we assume that the predicates assign the variables in the same order, i.e. the first parameter of the predicate is assigned as the first variable that appears in the purpose formula and so on. In the above example, the second parameter for both of these predicates is mapped to the variable x in ψ_3.

The process of evaluating the purpose predicates is an extended version of what we discussed in Section 4.2. Note that the discussion here is only for one parameter but the case for more parameters is similar. The meaning of $\texttt{Purpose}_{\psi}(\texttt{T}_i, \texttt{X}_i)$ is decided by grounding the purpose predicates and running the model-checking algorithm as discussed in [11].

- Using the $\texttt{W} = \texttt{WF-Of}(\texttt{T})$ parameter assertion, the workflow definition W to which T belongs can be extracted. The actual workflow definition for W in the system includes the definition of the hierarchy net $\mathcal{H}$ which gives the structure of the HN and other tasks belonging to this workflow.

- We define the mapping L' which assigns ground purpose assertions to each task. For each task T_i belonging to $\mathcal{T}_{\mathcal{H}}$, each parametrized vocabulary term P_v, and each X_i for which there is an assertion $\texttt{Attr}_v(\texttt{T}_i, \texttt{X}_i)$, we assign the assertion $P_v(X_i)$ or $\neg P_v(X_i)$ to task T_i depending on whether $\texttt{Attr}_v(\texttt{T}_i, \texttt{X}_i)$ is the case or $\neg\texttt{Attr}_v(\texttt{T}_i, \texttt{X}_i)$, i.e.:

$$\langle P_v(X_i), T_i\rangle \in L' \text{ iff } \texttt{Attr}_v(\texttt{T}_i, \texttt{X}_i)$$

- For each assertion $P_v(X_i)$ (which is ground) we introduce an atomic proposition $p_v^{x_i}$. The truth value of this proposition is decided for each task based on the value of $P_v(X_i)$ for that task, i.e:

$$\langle p_v^{x_i}, T_i\rangle \in L \text{ iff } \langle P_v(X_i), T_i\rangle \in L'$$

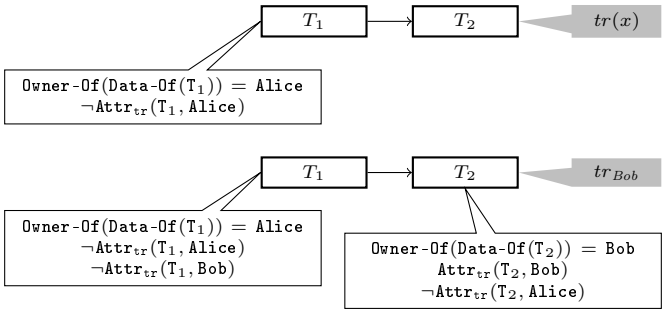

Figure 6.1: A very simple workflow with a parametrized purpose before (top) and after (bottom) instantiation of task T_2. The shaded callout shows how the purpose predicate is converted to an atomic proposition by instantiating its variable.

- Now, considering $\texttt{Purpose}_{\psi(x)}(\texttt{T}_i, \texttt{X}_i)$, we replace all occurrences of the variable x with X_i[1] and for each purpose predicate P_v and each X_i, we replace each occurrence of the $P_v(X_i)$ (which is ground) with an atomic proposition $p_v^{x_i}$. The formula $\psi(X_i)$ converted in this way no longer contains any variable.
- Now we run the purpose model-checking algorithm on $\psi(X_i)$ in the context of the hierarchy net $\mathcal{H}$ to which T_i belongs and the labelling function L constructed as discussed above. If $T_i \in [\![\psi(X_i)]\!]_{\mathcal{H},L}$ we will have $\texttt{Purpose}_{\psi}(\texttt{T}, X_i) \equiv \top$, otherwise $\texttt{Purpose}_{\psi}(\texttt{T}, X_i) \equiv \bot$.

As we see in the above, we create a new atomic proposition for each permutation of values for x_i's in the purpose assertion $P_v(x_1, \cdots, x_n)$. Since in our applications we are always dealing with a finite number of concrete objects and finite workflows this is always finite but with large numbers of objects, the number of propositions may be very large.

If we limit the policies to one workflow instance and rule out cross-workflow and cross-workflow instance policies similar to what we did in Section 5.1, the variables will range over the objects assigned to the workflow instance and we can benefit from the fact that the number of objects assigned to a single workflow instance is usually very small, since workflow instances are often *case-driven* and focus on one or a few *cases* [1]. Moreover, there are not a lot of instances of purposes that are truly parameterized and even those that are parameterized are most of the times unary.

Recall the example policy above, which can be re-written as:

$$\phi = \forall^{TASKI} t\ \forall^{PAT} p\ \forall^{PAT} q\ \ (\texttt{Owner-Of}(\texttt{Data-Of}(t)) = p \wedge p \neq q) \rightarrow \neg\texttt{Purpose}_{\psi_3(x)}(\texttt{Task-Type-Of}(t), q)$$

in which $\psi_3(x) \equiv \langle \mathbf{A} \rangle \mathrm{tr}(x) \vee \langle \mathbb{F} \rangle \mathrm{tr}(x)$.

Now, as shown in Figure 6.1, consider a very simple workflow with two consecutive tasks T_1 and T_2 and assume in an instance of this workflow, Alice's record is already assigned to task T_1 and a request to instantiate T_2 wants to assign Bob's record. Moreover, assume that $tr(x)$ which denotes treatment of patient x is assigned to T_2. The following assertion, which we call A are, thus, part of this assertions for the would-be system state for this request:

$$A \equiv \texttt{Task-Type-Of}(\texttt{K}_1) = \texttt{T}_1 \wedge \texttt{Task-Type-Of}(\texttt{K}_2) = \texttt{T}_2 \wedge$$
$$\texttt{Owner-Of}(\texttt{Data-Of}(\texttt{K}_1)) = \texttt{Alice} \wedge \texttt{Owner-Of}(\texttt{Data-Of}(\texttt{K}_2)) = \texttt{Bob} \wedge$$
$$\neg\texttt{Attr}_{tr}(\texttt{T}_1, \texttt{Bob}) \wedge \neg\texttt{Attr}_{tr}(\texttt{T}_1, \texttt{Alice}) \wedge \texttt{Attr}_{tr}(\texttt{T}_2, \texttt{Bob}) \wedge \neg\texttt{Attr}_{tr}(\texttt{T}_2, \texttt{Alice})$$

[1]Note that although these are well-intended, technically, the resulting formula after replacing the variable with a constant symbol does not belong to the language we defined, since X_i is now actually a constant symbol in the purpose language but we did not define constants for this language. We consider these intermediary processing forms and disregard this technical issue here for the sake of simplicity.

From $A \wedge \phi$ it follows that:

$$\neg\texttt{Purpose}_{\psi_3(x)}(\texttt{Task-Type-Of}(\texttt{K}_1), \texttt{Bob}) \wedge$$
$$\neg\texttt{Purpose}_{\psi_3(x)}(\texttt{Task-Type-Of}(\texttt{K}_2), \texttt{Alice})$$

Or: $\neg\texttt{Purpose}_{\psi_3(x)}(\texttt{T}_1, \texttt{Bob}) \wedge \neg\texttt{Purpose}_{\psi_3(x)}(\texttt{T}_2, \texttt{Alice})$. To evaluate $\texttt{Purpose}_{\psi_3(x)}$, we do as follows:

- From the task attribute assertions (last line of A above) the atomic propositions tr_{Alice} and tr_{Bob} are created and the labelling function L is constructed as $\{(tr_{Bob}, T_2)\}$.
- To evaluate $\neg\texttt{Purpose}_{\psi_3(x)}(\texttt{T}_1, \texttt{Bob})$, we form $\psi_3(Bob) \equiv \langle \mathbf{A} \rangle tr_{Bob} \vee \langle \mathbb{F} \rangle tr_{Bob}$. Now, we run the purpose model-checker on $\psi_3(Bob)$ in the context of $\mathcal{H}_1$ the HN for this simple workflow and the labelling function L created above. The result shows that $T_1 \in [\![\psi_3(Bob)]\!]_{\mathcal{H}_1, L}$, thus, $\texttt{Purpose}_{\psi_3(x)}(\texttt{T}_1, \texttt{Bob}) \equiv \top$, and hence $\neg\texttt{Purpose}_{\psi_3(x)}(\texttt{T}_1, \texttt{Bob}) \equiv \bot$.

Therefore, since $A \wedge \phi \equiv \bot$ and the access is rejected.

Bibliography

[1] W. M. P. Aalst, *Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management*, Lectures on Concurrency and Petri Nets, vol. 3098, Springer Berlin Heidelberg, 2004, pp. 1–65.

[2] E. Bertino, E. Ferrari, and V. Atluri, *The Specification and Enforcement of Authorization Constraints in Workflow Management Systems*, ACM Trans. Inf. Syst. Secur. **2** (1999), no. 1, 65–104.

[3] J. W. Byun, E. Bertino, and N. Li, *Purpose-Based Access Control of Complex Data for Privacy Protection*, SACMAT '05: Proceedings of the 10th ACM Symposium on Access Control Models and Technologies, ACM, 2005, pp. 102–110.

[4] J. Crampton, *A Reference Monitor for Workflow Systems with Constrained Task Execution*, SACMAT'05: Proceedings of the 10th ACM Symposium on Access Control Models and Technologies, 2005, pp. 38–47.

[5] J. Crampton and H. Khambhammettu, *On delegation and workflow execution models*, SAC'08: Proceedings of the 2008 ACM Symposium on Applied Computing, 2008, pp. 2137–2144.

[6] *The Enterprise Privacy Authorization Language (EPAL 1.1)*, `http://www.zurich.ibm.com/security/enterprise-privacy/epal`, 2003.

[7] Jean H. Gallier, *Logic for Computer Science: Foundations of Automatic Theorem Proving*, Harper & Row Publishers, Inc., New York, NY, USA, 1985.

[8] Joseph Y. Halpern and Vicky Weissman, *Using first-order logic to reason about policies*, ACM Trans. Inf. Syst. Secur. **11** (2008), no. 4, 21:1–21:41.

[9] M. Hilty, D. Basin, and A. Pretschner, *On Obligations*, ESORICS 2005: Proceedings of the 10th European Symposium On Research in Computer Security (Milan, Italy), September 2005, pp. 98–117.

[10] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems*, Cambridge University Press, 2004.

[11] M. Jafari, P. W. L. Fong, R. Safavi-Naini, and K. Barker, *A Framework for Expressing and Enforcing Purpose-Based Privacy Policies*, Tech. report, University of Calgary, 2013-1037-04, 2013.

[12] *NCI Thesaurus v.12.04e*, 2012.

[13] *RDF/XML Syntax Specification (Revised)*, `http://www.w3.org/TR/rdf-syntax-grammar/`, 2004.

[14] *SNOMED CT, Systematized Nomenclature of Medicine-Clinical Terms*, `http://www.ihtsdo.org/snomed-ct`, 2012.

[15] *SPARQL 1.1 Query Language*, `http://www.w3.org/TR/2013/REC-sparql11-query-20130321`, 2013.

[16] *Workflow Management Coalition Terminology and Glossary*, Tech. Report WFMC-TC-1011, 1999.

[17] *eXtensible Access Control Markup Language (XACML) Version 3.0*, http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf, 2013.

[18] *Privacy policy profile of XACML v2.0*, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-privacy_profile-spec-os.pdf, 2005.

[19] *XML Path Language (XPath) Version 1.0*, http://www.w3.org/TR/xpath/, 1999.

Appendix A

An Access Control Policy Language Based on Many-Sorted First-Order Logic

In order to abstract away the specifics of the access control system, we use a very general language for modelling access control policies based on *many-sorted first order logic* which is an extension of first-order logic in which variables have *types* and similarly, functions and predicates have a type signature that specifies the inputs and output *types*. We briefly discuss this language here; for a detailed account of the formal syntax and semantics for many-sorted first order logic see [7].

Types: Assume Y is the set of *types*. For example, *TASK* represents the type for *tasks*.

Variables: There is a set of variables corresponding to each type y which is shown as V_y. For example, $V_{TASK} = \{t\}$ indicates that the there is only one variable of type *TASK* which is shown as t.

Functions Signature Map: Assuming that *Fu* is the set of all functions, $\mathcal{S}_F : Fu \mapsto Y^* \times Y$ [1], is called the *functions signature map* and assigns a type signature (u, y) to each function f, in which $u \in Y^*$ specifies the arity and types of the inputs of the function and $y \in Y$ is the output type. For example, the function `UserOf` which returns the user assigned to a task has the signature $(TASK, USER)$, i.e. it has one input of type *TASK* and its output is of type *USER*.

Predicates Signature Map: Similarly, assuming *Pr* is the of all predicates, $\mathcal{S}_P : Pr \mapsto Y^*$ is called the *predicates signature map* and assigns a type signature $u \in Y^*$ to each predicate p specifying its arity and types of the inputs. For example, the predicate *Admin* which is true if the user has the administrator role has the signature *USER*, i.e. it has one input of type *USER*.

A.1 Syntax

The syntax of the language is defined similar to the standard first-order predicate logic. We assume t represents any variable of type *TASK*, x represents any variable (including t), f represents any n-ary function, and P represents any n-ary predicate ($n \geq 0$):

$$\tau ::= x \mid f(x, \cdots, x)$$
$$\phi ::= \top \mid P(\tau, \cdots, \tau) \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\forall^y x\phi) \mid (\exists^y x\phi)$$

We call the components created by the first rule (τ) a *term*. Note that a precise definition of the syntax must ensure that *types* for variables, functions and predicates are according to their signatures (see [7]).

[1] Y^* is the set of strings of size zero or higher, made of the symbols belonging to the set Y.

A.2 Semantics

The semantics for this language is very similar to standard first-order logic, based on the notion of a *model*, shown as $\mathcal{M}$ which includes the following components:

- A number of non-empty sets A_y representing the set of concrete objects pertaining to each type y, called the interpretation for type y.
- A *function interpretation* for each n-ary function f defined as $f^{\mathcal{M}} : A_{y_1} \times \cdots \times A_{y_n} \mapsto A_{y_{n+1}}$ in which y_i's accord with the signature of f.
- A *predicate interpretation* for each n-ary predicate P defined as $P^{\mathcal{M}} \subseteq A_{y_1} \times \cdots \times A_{y_n} \mapsto \{true, false\}$ in which y_i's accord with the signature of P. The interpretation of each predicate assigns a truth value to each permutation of concrete values with suitable artiy and type.

Moreover, a look-up table shown as E is assumed which maps each variable x_i of type y_i to a concrete object from A_{y_i}. We show a lookup table in which variable x is mapped to object a as $E_{[x \mapsto a]}$ and the value assigned to x in E as x^E.

On this basis, the semantic of the language is defined by the *satisfaction relation* ($\models$) in the context of the look-up table. $\mathcal{M} \models_E \phi$ denotes that the model $\mathcal{M}$ satisfies the formula ϕ in the context of the look-up table E, and $\mathcal{M} \not\models_E \phi$ denotes that is not the case.

The semantics for terms is defined by inductively defining their value in the context of the model $\mathcal{M}$ and the look-up table E:

- The value of each variable x is defined as x^E.
- The value of each n-ary function $f(x_1, \cdots, x_n)$ is defined as $f^{\mathcal{M}}(x_1, \cdots, x_n)$.

Accordingly, the satisfaction relation is defined inductively as below:

$\mathcal{M} \models_E \top$.
$\mathcal{M} \models_E (\neg\phi)$, iff $\mathcal{M} \not\models_E \phi$.
$\mathcal{M} \models_E P(x_1^E, \cdots, x_n^E)$, iff $(x_1^E, \cdots, x_n^E, true) \in P^{\mathcal{M}}$.
$\mathcal{M} \models_E (\phi_1 \wedge \phi_2)$, iff $\mathcal{M} \models_E \phi_1$ and $\mathcal{M} \models_E \phi_2$.
$\mathcal{M} \models_E (\forall^y\ x\phi)$, iff for all $a \in A_y$ it holds that: $\mathcal{M} \models_{E[x \mapsto a]} \phi$ (where y is the type of x).
$\mathcal{M} \models_E (\exists^y\ x\phi)$, iff for some $a \in A_y$ it holds that: $\mathcal{M} \models_{E[x \mapsto a]} \phi$ (where y is the type of x).

We also define the following as derived forms:

$\mathcal{M} \models_E \bot$, iff $\mathcal{M} \models_E \neg(\top)$.
$\mathcal{M} \models_E (\phi_1 \vee \phi_2)$, iff $\mathcal{M} \models_E \neg(\neg\phi_1 \wedge \neg\phi_2)$.
$\mathcal{M} \models_E (\phi_1 \rightarrow \phi_2)$, iff $\mathcal{M} \models_E \neg(\phi_1 \wedge \neg\phi_2)$.