A GENERAL FRAMEWORK FOR WEB CONTENT FILTERING

by Elisa Bertino, Elena Ferrari, Andrea Perego

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

A General Framework for Web Content Filtering

Elisa Bertino
CS & ECE Departments, Purdue University, IN, USA
bertino@cerias.purdue.edu

Elena Ferrari and Andrea Perego
DICOM, Università Degli Studi dell'Insubria, Varese, Italy
{elena.ferrari, andrea.perego}@uninsubria.it

Abstract

Web content filtering is a means to make an end user aware of the 'quality' of Web resources by evaluating their content and/or characteristics against his/her preferences. Although it can be used for a variety of purposes, currently it is mainly enforced by a number of Web applications as a service for parental control purposes, and for regulating the access to Web content of users connected to the networks of enterprises, libraries, schools, etc. This is obtained by integrating and optimizing established techniques in the fields of, e.g., data mining and firewall blocking, in order to provide a service addressing the needs of very specific user categories. Yet, what is lacking is a unified filtering framework for all the possible application domains, making it possible to enforce interoperability among the different filtering approaches and the systems based on them.

In this paper, a *multi-strategy* approach is described, which integrates the available techniques and focuses on the use of metadata for rating and filtering Web information. Such an approach consists of a filtering model, referred to as MFM, which provides a general representation of the Web content filtering domain, independently from its possible applications, and of two prototype implementations, partially carried out in the framework of the EU projects EUFORBIA and QUATRO, and designed for different application domains: users' protection and Web quality assurance, respectively.

Besides modelling the characteristics of the filtering domain, MFM improves the approaches currently available by supporting policies taking into account both users' and resources' characteristics, described by using multiple metadata vocabularies. Another novel feature of MFM is the support for *supervised filtering*, according to which policies are specified by *supervisors*, with different authority levels, who are in charge of determining the filtering rules to be applied to a set of users. As a result, MFM, on one hand, allows the enforcement of interoperability among filtering systems based on different approaches, and, on the other hand, it can be easily tailored to users' requirements and preferences.

1 Introduction

Web content filtering, i.e., the evaluation of Web resources against a given set of parameters, aims at making users aware of the content/characteristics of the data and services they are accessing. Such practice started in the 1990s, as soon as the Web became potentially accessible by everyone, since the lack of control over the publishing of Web content required tools for preventing specific categories of users (e.g., minors) from accessing inappropriate, or even harmful, content (e.g., pornography, violence, racism).

For this purpose, two main strategies have been adopted so far. The former is based on *white/black lists* of resources, which are classified as appropriate or inappropriate, respectively. Such classification is performed by rating agencies, which make then available to end users services, usually run by a proxy, in charge of performing filtering. By contrast, the latter approach is based on the evaluation of *content labels*, which are formal description of the content/characteristics of the resource with which they are associated. Such strategies has been established by the PICS W3C standard [14], which defines a general format for rating vocabularies and content labels. Labels are generated according to either a third-party or a self-rating approach. For instance, ICRA (`www.icra.org`) provides an online form, which allows Web site owners to automatically generate content labels. Filtering is performed by browser extensions, allowing users to specify preferences with respect to the type of resources they consider as inappropriate. Currently, both MS Internet Explorer and Netscape Navigator have PICS built-in support.

The PICS-based approach has been devised in order to address the main drawbacks of list-based filtering. In fact, labels *describe* resources' content/characteristics instead of stating whether they are in/appropriate. As a consequence, labels potentially allows a precise and accurate filtering. Moreover, the in/appropriateness of a resource is not determined by a rating service, but by end users' preferences. The available PICS-complaint rating vocabularies have been designed with the primary goal of simplifying the task of describing resources, and they consist of plain sets of descriptors, whereas none of them makes use of more sophisticated knowledge representation tools, such as conceptual hierarchies and ontologies. Nonetheless, content labels are currently the more effective way of enforcing Web content filtering.

Besides users' protection, another purpose for which Web sites are rated concerns ensuring the 'quality' of resources with respect to given requirements. In this scenario, trustmark agencies (also referred to as *certification authorities*) rate Web sites after evaluating their characteristics with respect to a set of rules. As an example, IQUA (`www.iqua.net`) verifies the reliability and authoritativeness of Web sites based, for instance, on the fact that they do not carry out or encourage illegal practices, they provide truthful information and do not use misleading advertising. Another example, not concerning the 'quality' of the information itself but of how it is represented, is provided by Segala (`www.segalamtest.com`), which verifies the accessibility of a Web site from mobile devices. Whenever a Web site satisfies the requirements defined by a trustmark agency, an icon is inserted in its pages as a certification mark.

Despite trustmark agencies adopt an approach very similar to that used for rating Web sites for users' protection purposes, content labels are not used, and no software tools are available for performing filtering. By contrast, it would be very simple to associate with resources metadata describing their 'quality', since the set of evaluation criteria defined by trustmark agencies can be represented as descriptor vocabularies. This would allow not only the development of tools able to notify the presence/absence of a trustmark, but it would also permit users to specify their own 'quality' criteria. For instance, a user may decide that the quality of a Web site is determined by only a subset of the requirements defined by a given trustmark agency, or by requirements stated by two or more trustmark agencies.

The use of content labels would have the further advantage of providing the basis for the definition of a Web filtering platform common to any application. For this purpose, it is necessary a standardization effort of the structure of vocabularies and content labels. This would not necessary allow the enforcement of semantic interoperability, but it would provide Web users the possibility to access different types of metadata by using the same software tools.

This paper proposes extensions to current Web filtering practices, with the aim of overcoming their drawbacks. In particular, a *multi-strategy* approach is described, which integrates the available techniques and focuses on the use of metadata for rating and filtering Web information. Such an approach consists of a filtering model, referred to as MFM, which provides a general representation of the Web content filtering domain, independently from its possible applications, and of two prototype implementations, partially carried out in the framework of two EU projects (namely, EUFORBIA and QUATRO), and designed for different application domains: users' protection and Web quality assurance, respectively. Besides modeling the characteristics of the filtering domain, MFM improves the approaches currently available by supporting policies taking into account both users' and resources' characteristics, described by using multiple metadata vocabularies. Another novel feature of MFM is the support for *supervised filtering*, according to which policies are specified by *supervisors*, with different authority levels, who are in charge of determining the filtering rules to be applied to a set of users. As a result, MFM, on one hand, allows the enforcement of interoperability among filtering systems based on different approaches, and, on the other hand, it can be easily tailored to users' requirements and preferences.

The remainder of this paper is organized as follows. Section 2 illustrates the proposed filtering model, describing its main components, whereas Section 3 explains how policies are specified and enforced. Section 4 describes the two different implementation of MFM in the framework of the EU projects EUFORBIA and QUATRO, concerning the application domains of Web users' protection and Web quality assurance, respectively. Finally, Section 5 discusses related work, whereas Section 6 concludes the paper and outlines future research directions.

Symbol		Notion
Set	Element	
$\mathcal{A}$	a	Agent attribute
$\mathcal{AN}$	an	Agent attribute name
$\mathcal{AD}$	ad	Agent attribute domain
$\mathcal{AT}$	at	Agent attribute type
$\mathcal{AV}$	av	Agent attribute value
$\mathcal{C}$	cls	Agent class
$\mathcal{CI}$	cls_id	Agent class identifier
$\mathcal{IN}$	$inst$	Agent class instance
$\mathcal{CH}$	cls	Agent class hierarchy
$\mathcal{AG}$	ag	Agent
$\mathcal{AGI}$	ag_id	Agent identifier
$Pred$	$pred$	CSL predicate
$\mathcal{EX}$	ex	CSL expression
$\mathcal{AS}$	ag_spec	Agent specification
$\mathcal{SVSB}$	sup_spec	Supervision specification
$\mathcal{FPB}$	fp	Filtering policy

Table 1: Notations used in Section 2

2 MFM: A Multi-strategy Filtering Model

The general filtering model described in this section, referred to as MFM, provides a general framework to denote interaction constraints between two sets of entities (referred to as *agents*) in a given domain. MFM is not designed for a particular context, and cannot be *directly* applied. Rather, its aim is to define a basic data structure which can be used to generate *instances*, which customize their characteristics according to the application domain.

In this section, after having provided a general overview of the model, we introduce the notion of *agent* (subsuming those of supervisors, subjects, and objects), and we describe how agents are characterized by associating with them *agent classes*, organized into a hierarchy. Then, we illustrate the MFM constraint specification language and the notion of *policy*

The notations used in this section are reported in Table 1.

2.1 Overview

MFM consists of three main components:

- a set of constructs to denote agents' identities and characteristics;
- a policy specification language;
- a set of rules for policy propagation and conflict resolution.

In MFM, agents can be considered as entities which can perform and/or are subject to a given set of operations. Depending on whether they have an *active* or *passive* role in the interaction process, they can be grouped into two

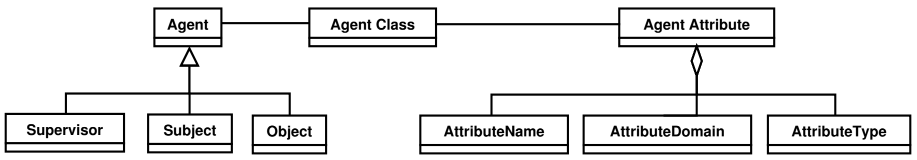


Figure 1: UML diagram of MFM agents and agent classes

subsets: the sets of *subjects* and *objects*, respectively. Examples of agents in the filtering domain are, for instance, users and Web pages, which perform the roles of subjects and objects, respectively, in an interaction process according to which users "access" Web pages.

Policies are rules specified by a class of agents, referred to as *supervisors*, stating which kind of operations subjects can or cannot perform on a given set of objects. They are denoted by the identifier of the supervisor who specified them, by a pair of subject and object sets, and by the *action* to be performed by the system. An action consists of two components: an *operation*, determining the type of interaction (e.g., "access"), and a *sign*, which states whether an operation can ("+") or cannot ("−") be performed. Agent sets can be specified either explicitly, by listing the agents they contain (e.g., users u_1, u_2, and u_3 cannot access Web pages wp_1 and wp_2), or implicitly, by stating properties that agents must satisfy (e.g., "users who are students, and whose age is less than 16, cannot access Web pages regarding sexual content").

In order to describe agent properties, MFM adopts an object-oriented approach relying on the notion of *agent class*. An agent class (*class* for short) specifies a set of *attributes*, denoting agent characteristics relevant in a given domain (e.g., the age of a user, the content of a Web page). Agents are then associated with *class instances*, which are sets of *attribute-value* pairs (which can be either optional of mandatory) denoting agent properties. Finally, agent classes are organized into *class hierarchies*, which are exploited by a *policy propagation* principle according to which a policy applying to an agent class is inherited by all its children. Figure 1 depicts an UML representation of the notions of agents and agent classes.

Policies are associated with a *sign*, according to which they can be either *positive* or *negative*. For instance, consider a policy concerning a set of users and a set of Web sites: if it is positive, the users can access the Web sites; if it is negative, they cannot. Finally, since the use of this feature may result in the specification of *conflicting* policies (i.e., policies on the same pair of agent sets and of the same type, but with different sign), a *conflict resolution mechanism* is provided in order to determine which of them is prevailing.

MFM can support multiple class hierarchies, of which a standard, hierarchical, representation is given. This is obtained by defining a general root class $\top_{\mathcal{C}}$, to which the root classes of the supported vocabularies (referred to as *class sub-hierarchies*) are assigned as direct children. As a result, MFM is independent

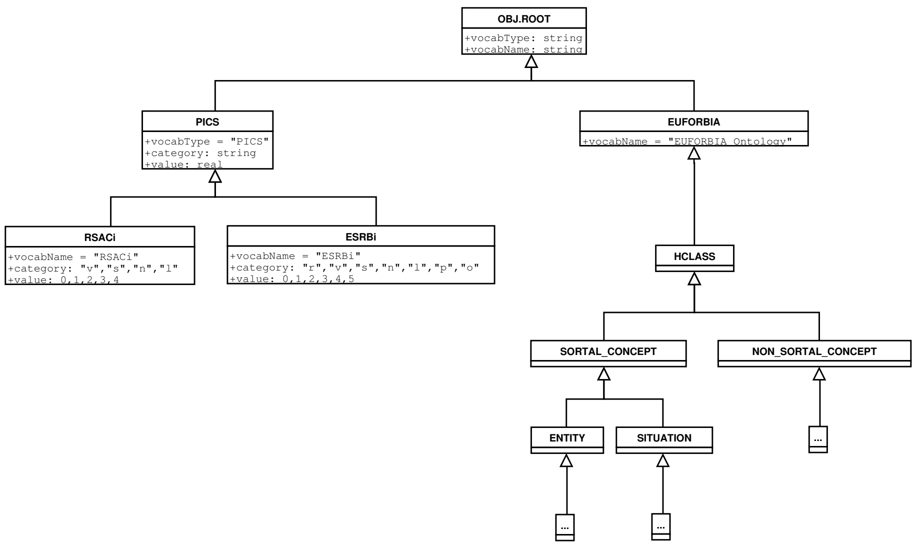


Figure 2: An example of class hierarchy merging multiple metadata vocabularies

from the metadata vocabularies used to characterize agents. As an example, Figure 2 depicts an agent class hierarchy, merging two PICS-based rating systems (namely, RSAC*i* and ESRB*i*[1]), and the conceptual hierarchy developed in the framework of the project EUFORBIA (see Section 4.1). For sake of clarity, in the figure only the upper levels of the tree are reproduced.

Note, however, that such metadata vocabulary integration totally differs from approaches, as the ABC-based one [12], aiming to provide semantic interoperability among ontologies. The objective is to harmonize only their structure, in order to easily specify policies ranging over different class hierarchies.

2.2 Agents

In MFM, agents are denoted by an identifier and characterized by associating with them one or more *class instances.* Agent attributes are denoted by a name *an*, which univocally identifies them, by a domain *ad*, and by a type *at*, determining whether they are optional or mandatory. Class instances sharing the same attributes are then grouped into *classes.* Finally, classes are organized into *class hierarchies*, according to which the set of attributes associated with a given class is inherited by all its children.

[1] These rating systems have been chosen since their simple structure is more suitable for examples. RSAC*i* (Recreational Software Advisory Council on the internet: `http://www.rsac.org`) is the predecessor of ICRA, and it is no longer available; ESRB*i* (Entertainment Software Rating Board Interactive: `http://www.esrb.org`) is a non-profit organization releasing ratings for entertainment software.

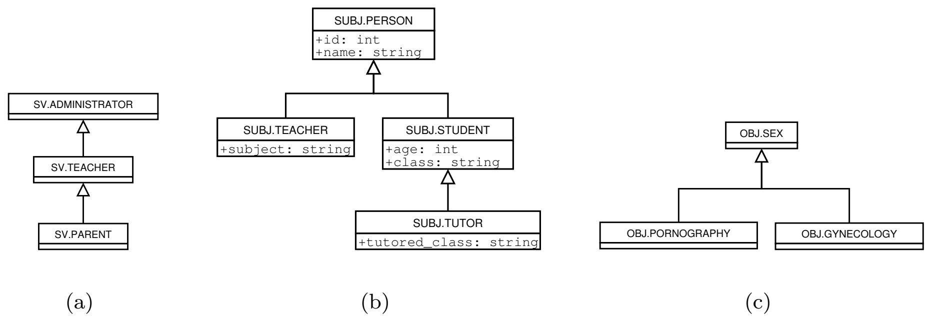


Figure 3: Examples of: (3(a)) supervisor, (3(b)) subject, and (3(c)) object class hierarchies

The notions of agent class and agent class instance are formally defined as follows.

Definition 2.1 (Agent Class) An agent class $cls \in \mathcal{C}$ is a tuple $(cls\_id, attr, parent\_id)$, where $cls\_id \in \mathcal{CI}$ is the class identifier, $attr \subseteq \mathcal{A}$ is the set of attributes characterizing class *cls*, and *parent_id* is the identifier of the parent of *cls*, or a null value if *cls* is the root class. □

Definition 2.2 (Agent Class Instance) A class instance $inst \in \mathcal{IN}$ is a pair $(cls\_id, state)$ where $cls\_id \in \mathcal{CI}$ is the identifier of class *cls*, of which *inst* is an instance, and *state* is a set, possibly empty, of attribute-value pairs $\{an_1 : av_1, \ldots, an_n : av_n\}$. □

We can now formally define the notion of agent:

Definition 2.3 (Agent) An agent $ag \in \mathcal{AG}$ is represented by a pair $(ag\_id, prop)$, where $ag\_id \in \mathcal{AGI}$ is the agent identifier, and *prop* is a set of class instances $IN \subseteq \mathcal{IN}$, possibly empty, denoting the properties of agent *ag*. □

Depending on their role, agents are grouped into three different subsets, the set $\mathcal{SV}$ of *supervisors*, the set $\mathcal{S}$ of *subjects*, and the set $\mathcal{O}$ of *objects*. Similarly, the agent classes in $\mathcal{C}$ are organized into three distinct inheritance hierarchies, namely, the *supervisor*, *subject*, and *object class hierarchies* (denoted, respectively, $\mathcal{CH}_{\mathcal{SV}}$, $\mathcal{CH}_{\mathcal{S}}$, and $\mathcal{CH}_{\mathcal{O}}$). The role of an agent is then determined by the class hierarchy used for characterizing it. Note, however, that the same agent may be characterized by using more class hierarchies—i.e., it may have several roles (e.g., an agent may be both a supervisor and a subject).

As an example, Figure 3 shows three simple supervisor, subject, and object class hierarchies, each consisting of a single vocabulary.

Example 2.1 (Agents) Consider the supervisor, subject, and object class hierarchies in Figure 3. The following are examples of agents:

- (`John`, $\{$(SUBJ.PERSON, $\{$`id : John`, `name : John Brown`$\}$), (SV.ADMINISTRATOR, $\emptyset$)$\}$): it denotes an agent, whose identifier is `John`, associated with the subject class PERSON and with the supervisor class ADMINISTRATOR;
- (`Bob`, $\{$(SUBJ.STUDENT, $\{$`id : Bob`, `name : Bob Smith`, `age : 15`, `class : A2`$\}$)$\}$): it denotes an agent, whose identifier is `Bob`, associated with the subject class STUDENT; as can be seen, since SUBJ.STUDENT is child of SUBJ.PERSON, it inherites also its attributes;
- (`www.example.org`, $\{$(OBJ.SEX, $\emptyset$)$\}$): it denotes an agent, the identifier of which is `www.example.org`, associated with the object class SEX;
- (`www.somesite.net`, $\{$(OBJ.GYNECOLOGY, $\emptyset$)$\}$): it denotes an agent, the identifier of which is `www.somesite.net`, associated with the object class GYNECOLOGY.

2.3 Agent Specifications

Agents can be grouped into classes either explicitly, by listing their identifiers, or implicitly, by specifying conditions on agent classes and/or attributes that agents must satisfy.

MFM provides a language for implicitly denoting agents. The *constraint specification language* (CSL for short) is formally defined as follows:

Definition 2.4 (Constraint Specification Language) The MFM constraint specification language is denoted by $\mathcal{L}_{\text{cs}} = Var_{\mathcal{AG}} \cup Pred \cup Com$ where:

- $Var_{\mathcal{AG}}$ is a set of variables, ranging over the set of agents $\mathcal{AG}$;
- $Pred$ is a set of predicates on $Var_{\mathcal{AG}}$;
- $Com \in \{\vee, \wedge\}$ is a set of Boolean operators to combine predicates in $Pred$. □

CSL is used to express constraints on classes (*class predicates*) and attributes (*attribute predicates*).

A class predicate denotes all the agents associated with a given class. For this purpose, for each class $cls \in \mathcal{C}$, a corresponding predicate symbol $cls()$ is defined in $Pred$. The set of class predicates is denoted $Pred_{\mathcal{C}}$.

Attribute predicates denote a) all the agents where the value of a given attribute is equal to (different from, greater than, less than, etc.) the value of another attribute, or b) all the agents where a given attribute has a given (range of) value(s). Thus an attribute predicate consists of an attribute name, a comparison operator, and either another attribute name or a value. The set of attribute predicates is denoted $Pred_{\mathcal{A}}$.

We can now formally define the notion of CSL predicate:

Definition 2.5 (CSL Predicate) The two different classes of CSL predicates are formally defined as follows:

Class Predicate If $X \in Var_{\mathcal{AG}}$, $cls \in \mathcal{C}$, then $cls(X)$ is a class predicate.

Attribute Predicate Attribute predicates have one of the following two forms:

- if $cls, cls' \in \mathcal{C}$, $X, X' \in Var_{\mathcal{AG}}$, $an, an' \in \mathcal{AN}$, and OP is a comparison operator, then $cls(X).an$ OP $cls'(X').an'$ is an attribute predicate;
- if $cls \in \mathcal{C}$, $X \in Var_{\mathcal{AG}}$, $an \in \mathcal{AN}$, $av \in \mathcal{AV}$, and OP is a comparison operator, then $cls(X).an$ OP av is an attribute predicate. □

Predicates of the same or different type can be combined by using the "$\vee$" and "$\wedge$" Boolean operators in order to increase the expressivity of the predicates which can be specified. This feature is enforced by the notion of *CSL expression*, formally defined as follows:

Definition 2.6 (CSL Expression) The set $\mathcal{EX}$ of CSL expressions is defined as follows:

Simple Expression Each element in $Pred = Pred_{\mathcal{C}} \cup Pred_{\mathcal{A}}$ is a CSL expression.

Compound Expression If ex_1 and ex_2 are CSL expressions, then $ex_1 \vee ex_2$ and $ex_1 \wedge ex_2$ are CSL expressions. □

As already stated, agent sets can be denoted either explicitly, by listing their identifiers, or implicitly, by specifying CSL expressions. These two possibilities are enforced by the notion of *agent specification*, formally defined as follows:

Definition 2.7 (Agent Specification) An agent specification $ag\_spec \in \mathcal{AS}$ has one of the following two forms:

Explicit Agent Specification Each set $AGI \in 2^{\mathcal{AGI}}$ of agent identifiers is an agent specification.

Implicit Agent Specification Each CSL expression $ex \in \mathcal{EX}$ is an agent specification. □

The sets of explicit and implicit agent specifications are denoted, respectively, by $\mathcal{AS}_{\mathcal{AGI}}$ and $\mathcal{AS}_{\mathcal{EX}}$. The set $\mathcal{AS}$ of agent specification can then be denoted $\mathcal{AS} = \mathcal{AS}_{\mathcal{AGI}} \cup \mathcal{AS}_{\mathcal{EX}}$.

In MFM, agent specification pairs are used for two purposes:

1. to denote the sets of subjects and objects to which a policy applies (*subject-object specification*);
2. to denote a set of supervisors and the corresponding set of supervised subjects (*supervision specification*).

The notion of subject-object specification (s-o specification, for short) is formally defined as follows.

Definition 2.8 (S-O Specification) An s-o specification so_spec is a pair $(subj\_spec, obj\_spec)$, where $subj\_spec$ is an agent specification denoting a set of subjects $S \subseteq \mathcal{S}$, and obj_spec is an agent specification denoting a set of objects $O \subseteq \mathcal{O}$. □

Example 2.2 (S-O Specifications) Consider the subject and object class hierarchies in Figure 3. The following are examples of s-o specifications:

- $(\textsc{subj.person}(X), \textsc{obj.sex}(X))$: specifies a pair of agent sets consisting, respectively, of all the agents associated with the subject class PERSON, and of all the agents associated with the object class SEX;
- $(\textsc{subj.student}(X).\texttt{age} > \texttt{14}, \textsc{obj.sex}(X))$: specifies a pair of agent sets consisting, respectively, of all the agents associated with the subject class STUDENT whose age is more than 14, and of all the agents associated with the object class SEX;
- $(\textsc{subj.teacher}(X) \vee \textsc{subj.tutor}(X), \textsc{obj.gynecology}(X) \wedge \textsc{obj.pornography}(X))$: specifies a pair of agent sets consisting, respectively, of all the agents associated with the subject class TEACHER or TUTOR, and of all the agents associated with the object classes GYNECOLOGY and PORNOGRAPHY;
- $(\textsc{subj.person}(X), \{\texttt{www.example.org}\})$: specifies a pair of agent sets consisting, respectively, of all the agents associated with the subject class PERSON and of the agent with identifier `www.example.org`;
- $(\{\texttt{Bob}\}, \textsc{obj.gynecology}(X))$: specifies a pair of agent sets consisting, respectively, of the agent with identifier `Bob`, and of all the agents associated with the object class GYNECOLOGY.

Similarly, the set of supervisors and the corresponding set of supervised subjects are determined by a pair of agent specifications, referred to as *supervision specification*, possibly of different type, where the former denotes a set of agents $SV \subseteq \mathcal{SV}$ with the role of *supervisor*, whereas the latter denotes a set of agents $S \subseteq \mathcal{S}$ with the role of *supervised subject.* The set of supervised subjects is denoted by $\mathcal{SVS} \in 2^{\mathcal{S}}$. Thus, the notions of s-o specification and supervision specification differ only in the types of agent sets (subjects and objects, in the former, supervisors and supervised subjects, in the latter).

The notion of supervision specification is formally defined as follows.

Definition 2.9 (Supervision Specification) A supervision specification *sup_spec* is a pair (*sv_spec*, *su_spec*), where *sv_spec* is an agent specification denoting a set of supervisors $SV \subseteq \mathcal{SV}$, and *su_spec* is an agent specification denoting a set of supervised subjects $SVS \subseteq \mathcal{SVS}$. □

Example 2.3 (Supervision Specifications) Consider the supervisor and subject class hierarchies in Figure 3. The following are examples of supervision specifications:

- $(\textsc{sv.administrator}(X), \textsc{subj.person}(X))$: specifies that the agents associated with the supervisor class ADMINISTRATOR are supervisors of the agents associated with the subject class PERSON;
- $(\textsc{sv.teacher}(X), \textsc{subj.student}(X))$: specifies that the agents associated with the supervisor class TEACHER are supervisors of the agent associated with the subject class STUDENT;
- $(\{\texttt{Jane}\}, \{\texttt{Bob}\})$: specifies that the agent with identifier `Jane` is supervisor of the agent with identifier `Bob`.

2.4 Filtering Policies

In MFM, a *filtering policy* states that a given set of subjects can or cannot perform a given action upon a given set of objects. The two involved agent sets, the set of subjects and the set of objects, respectively, are denoted by either explicit or implicit agent specifications, specified by a set of supervisors. The *action* component of a policy consists of an operation type and a sign. Such pair denotes the action the system should perform whenever one of the subjects is accessing one of the objects. For instance, an *allow* operation grants users the access to an object, if the sign is positive, whereas it prevents the access, if the sign is negative.

Also, operations may be similar to *permissions*, as used in the access control domain (e.g., “read”, “write”, “execute”, “append”, “update”, “delete”). In MFM a predefined set of operations is not specified, since they are implementation-dependent and may vary depending on the application domain. Nonetheless, whenever more than one operation is supported, they may be organized into a hierarchy $(\mathcal{OPT}, \prec)$, which determines the *stronger* operation. For instance, given two operations `read` and `write`, we can specify an operation hierarchy `read` $\prec$ `write`, stating that the `write` operation subsumes the `read` operation, since if a subject can modify an object, he/she must be able to read the information contained in it. We then say that the `write` operation is stronger than the `read` one (denoted `write` $>$ `read`).

The last component of a filtering policy is the *supervision mode*, which determines how supervision is enforced. MFM supports the following three supervision modes:

Strict Supervision In case of positive policies, a subject can perform the action in the policy on the requested resource only with the explicit consent of the supervisor who specified the policy itself. In other words, the supervisor is notified that a supervised subject is requesting access to a resource, and he/she is asked whether such subject can actually perform or not the action stated by the policy.

Normal Supervision It is the default mode: the type of action a subject can/-cannot perform on the requested resource is determined by the operation and sign specified in the policy. In other words, the normal supervision mode does not require any explicit intervention of the supervisor in order to apply a policy, as in the strict supervision mode.

Light Supervision In case of negative policies, it is up to the subject to decide whether to perform or not the action in the policy, thus overriding the policy sign. As in the strict supervision mode, supervisors can track the behaviour of subjects.

Unlike the normal supervision mode, which applies to any application domain, the strict and light supervision modes are designed for domains, such as users’ protection, where access to resources may be blocked in case they are considered as inappropriate. In such contexts, the strict supervision mode concerns subjects to whom a very restrictive access to resources should be applied (e.g., minors to be protected from harmful contents, or users who should be monitored due to bad behaviour). By contrast, the light supervision mode concerns subjects (such as teachers, in a school context) who do not need to be protected as 'standard' subjects (e.g., students), or it can be used in order to control the inappropriate use of network resources (e.g., employees connecting to entertainment Web sites during working hours). In fact, since also in this case supervisors can monitor the submitted access requests, they can modify filtering policies or the supervision mode of subjects in case of 'bad' behaviour.

Based on the purpose of each supervision mode, and also in order to solve possible conflicts, we order supervision modes such that the `strict` mode is considered as stronger than the `normal` mode, whereas the `normal` mode is considered as stronger than the `light` mode (denoted `strict` > `normal` > `light`).

We can now formally define the notion of filtering policy.

Definition 2.10 (Filtering Policy) A filtering policy *fp* is a tuple (*sv_id*, *so_spec*, *act*, *sv_mode*) where:

- *sv_id* $\in \mathcal{SVI}$ is the identifier of the supervisor who specified the policy;
- *so_spec* is a s-o specification denoting the sets of agents $S \subseteq \mathcal{S}$ and $O \subseteq \mathcal{O}$;
- *act* = (*op_type*, *sign*), where *op_type* $\in \mathcal{OPT}$ denotes the operation type, whereas *sign* $\in \{+, -\}$ is the policy sign;
- *sv_mode* $\in$ {`strict`, `normal`, `light`} is the supervision mode. □

Example 2.4 (Filtering Policies) Assume an application domain where two MFM operations are supported, namely, `allow` and `notify`, where the former is considered as the stronger operation (i.e., `allow` > `notify`). Depending on the policy sign, the `allow` operation determines whether the access to a resource should be granted ("+") or prevented ("−"), whereas the `notify` operation always grants the access to the requested resource, but it notifies to the end user whether such resource is appropriate ("+") or not ("−").

Suppose now that no user can access contents regarding the sexual domain, unless he/she is a teacher or a tutor. Moreover, suppose that students older than 14 are allowed to access contents regarding gynecology. Finally, suppose that there is a Web site `www.example.org`, which, even though it is associated with the object class SEX, is considered appropriate for all the users. These requirements are enforced by John, one of the administrators (who, according to Example 2.3, are supervisors of all the subjects), by specifying the following filtering policies:

- $fp_1 = (\texttt{John}, (\textsc{subj.person}(X), \textsc{obj.sex}(X)), (\texttt{allow}, -), \texttt{normal})$
- $fp_2 = (\texttt{John}, (\textsc{subj.teacher}(X) \vee \textsc{subj.tutor}(X), \textsc{obj.sex}(X)), (\texttt{allow}, +), \texttt{normal})$
- $fp_3 = (\texttt{John}, (\textsc{subj.student}(X).\texttt{age} > \texttt{14}, \textsc{obj.gynecology}(X)), (\texttt{allow}, +), \texttt{normal})$
- $fp_4 = (\texttt{John}, (\textsc{subj.person}(X), \{\texttt{www.example.org}\}, (\texttt{allow}, +), \texttt{normal})$

However, Ted, a teacher (and thus supervisor of all the students), does not agree with policy fp_3. Thus, he specifies the following policy:

$$fp_5 = (\texttt{Ted}, (\textsc{subj.student}(X).\texttt{age} > \texttt{14}, \textsc{obj.gynecology}(X)), (\texttt{allow}, +), \texttt{strict})$$

stating that 15-aged students can access content concerning gynecology only with his explicit consent.

Finally, Jane, parent (and supervisor) of a 15-aged student called Bob, agrees that her son can access content concerning gynecology (so, she agrees with policy fp_3), but she thinks that he should be informed that it is inappropriate for him. Consequently, Jane specifies the following policy:

$$fp_6 = (\texttt{Jane}, (\{\texttt{Bob}\}, \textsc{obj.gynecology}(X)), (\texttt{notify}, -), \texttt{normal})$$

3 Filtering Enforcement

In this section, we first illustrate the mechanisms supported by MFM to propagate policies along the subject and object hierarchies, and for determining the prevailing one among the set of policies applying to a given subject-object pair. Then, we describe the policy validation and optimization procedures, and how filtering is enforced.

3.1 Policy Propagation and Conflict Resolution

In MFM, the hierarchical structure into which classes are organized is exploited by a *policy propagation* mechanism according to which a predicate specified for a class *cls* applies to all its children. According to this principle, given two classes *cls*, $cls' \in \mathcal{C}$, with $cls' \prec_{\mathcal{CH}} cls$, a predicate on *cls* applies also to cls'.

This feature, along with the support for positive and negative policies, greatly increases the expressivity of the model and dramatically reduces the number of policies which must be specified, but, at the same time, it may lead to the specification of conflicting policies, that is, policies on the same agents but with different signs. Thus, MFM provides a *conflict resolution mechanism* allowing one to decide, among a set of conflicting policies, which of them is prevailing.

The MFM conflict resolution policy consists of 5 steps:

1. the *authority levels* of the supervisors who specified the conflicting policies are compared. If such supervisors are associated with different authority levels, the prevailing policy is the one specified by the supervisor with the higher authority level; otherwise,

2. the *specificity* of the conflicting policies is evaluated. If a policy is more specific with respect to the subject and/or object specification, it is considered as prevailing; otherwise,

3. the *operations* specified in the policies are compared. If they are different, the prevailing policy is the one specifying the stronger operation; otherwise,

4. the *signs* of the policies are compared. If they are different, the prevailing policy is the one with the stronger sign; otherwise,

5. the prevailing policy is the one with the stronger supervision mode.

The support for different authority levels addresses the need of taking into account that opinions of different supervisors about what is in/appropriate for a subject may have a different 'weight' (e.g., in a school context, if the opinions of parents are in conflict with those of teachers, the former should prevail). For this purpose, in MFM, the class hierarchy $\mathcal{CH}_{\mathcal{SV}}$ is used to determine the authority level of supervisors, according to the principle that supervisors associated with child classes have a stronger authority than those associated with parent classes. As an example, consider the supervisor hierarchy in Figure 3, which can be denoted PARENT $\prec$ TEACHER $\prec$ ADMINISTRATOR. In such a case, supervisors associated with class PARENT have stronger authority than those associated with class TEACHER, who, in turn, have stronger authority than those associated with class ADMINISTRATOR. Note, however, that a supervisor may be associated with more than one class. For instance, let us suppose that Jane (see Example 2.1) is both the mother (parent) of Bob and a teacher. In such a case, the authority of the supervisor is determined by the class denoting the stronger authority.

These rules are enforced by the notion of *stronger supervision authority*, which is formally defined as follows:

Definition 3.1 (Stronger Supervision Authority) Let sv_1, $sv_2 \in \mathcal{SV}$, be two supervisors, and let $C(sv_1)$ and $C(sv_2)$ be the sets of supervisor classes associated with them. We say that sv_1 is stronger than sv_2, denoted $sv_1 > sv_2$, iff $\forall cls \in C(sv_2)\ \exists cls' \in C(sv_1)$ such that $cls' \prec_{\mathcal{CH}_{\mathcal{SV}}} cls$. $\square$

The second step of the conflict resolution policy is based on the principle according to which most specific policies are *stronger* than less specific ones. In order to verify the specificity degree of a policy, we consider how subjects and objects are specified. More precisely, the strongest agent specification is the one denoting a subset of the agents denoted by the other ones; otherwise, agent specifications are considered equally strong. In other words, the strongest agent specification is the one *included* by the other ones. By using the notation adopted in Description Logics [5], we can then say that, given two agent specifications ag_spec, ag_spec', ag_spec is stronger than ag_spec' iff $ag\_spec \sqsubset ag\_spec'$.

For explicit agent specifications or agent specifications consisting of simple expressions (i.e., predicates), this applies in the following cases:

1. $ag\_spec, ag\_spec' \in \mathcal{AS}_{\mathcal{AGI}}$ and $ag\_spec \subset ag\_spec'$;

2. $ag\_spec \in \mathcal{AS}_{\mathcal{AGI}}$, $ag\_spec' \in Pred$, and ag_spec is a subset of the agents denoted by ag_spec';

3. $ag\_spec, ag\_spec' \in Pred$ are predicates over classes cls and cls', respectively, and $cls \prec_{\mathcal{CH}} cls'$;

4. $ag\_spec \in Pred_{\mathcal{A}}$ is an attribute predicate and $ag\_spec' \in Pred_{\mathcal{C}}$ is a class predicate, over the same agent class *cls*;

5. $ag\_spec, ag\_spec' \in Pred_{\mathcal{A}}$ are attribute predicates over the same agent class *cls*, and *ag_spec* denotes a subset of the agents denoted by *ag_spec'* (e.g., $ag_spec = $ SUBJ.STUDENT(X).`age` > 16 and $ag_spec' = $ SUBJ. STUDENT(X).`age` > 14).

Note that rule 3 applies also when predicates are of different type. Consequently, given two agent specification $ag_spec = $ SUBJ.STUDENT(X) and $ag_spec' = $ SUBJ.PERSON(X).`age` > 14, both applying to the same user, *ag_spec* is considered more specific than *ag_spec'* since SUBJ.STUDENT is a child of SUBJ.PERSON.

In case of agent specifications consisting of compound expressions, we consider their disjunctive normal form (DNF) in order to determine the most specific one. Note that any CSL expressions can be put in DNF without being modified, since our language supports only the "$\wedge$" and "$\vee$" Boolean operators. For instance, the expression $ex = pred_1 \wedge pred_2 \vee pred_3$ (where $pred_1, \ldots, pred_3 \in Pred$) can be normalized as $ex = (pred_1 \wedge pred_2) \vee (pred_3)$.

Let us first consider how we determine the more specific expression between two conjunctions. Given two conjunctions $ex_{\wedge} = (pred_1 \wedge pred_2)$ and $ex'_{\wedge} = (pred'_1 \wedge pred'_2)$, we say that $ex_{\wedge}$ is more specific than $ex'_{\wedge}$ ($ex_{\wedge} \sqsubset ex'_{\wedge}$) if one of the following conditions holds:

- $pred_1 \sqsubset pred'_1$ and $pred_2 \sqsubset pred'_2$;
- $pred_1 \sqsubset pred'_2$ and $pred_2 \sqsubset pred'_1$.

In case one of the expressions to be compared is a simple one, the corresponding predicate must be included by all the conjuncts of the other expression in order to be considered more specific. Thus, the expression $ex'' = pred''$ is more specific than $ex'_{\wedge}$ iff $pred'' \sqsubset pred'_1$ and $pred'' \sqsubset pred'_2$.

The more specific between two agent specifications in DNF is then determined by comparing their disjuncts according to the rules above. More precisely, given two agent specifications in DNF $ex_{\vee} = \bigvee_{i=1}^{n} ex_i$ and $ex'_{\vee} = \bigvee_{j=1}^{m} ex'_j$, we say that $ex_{\vee}$ is more specific than $ex'_{\vee}$ ($ex_{\vee} \sqsubset ex'_{\vee}$) iff for each disjunct ex_i in $ex_{\vee}$, there exists at least a disjunct ex'_j in $ex'_{\vee}$ such that $ex_i \sqsubset ex'_j$.

This applies also when we must compare an implicit agent specification with an explicit one. In fact, an explicit agent specification $ag\_spec = \{ag\_id_1, \ldots, ag\_id_n\}$ can be also expressed as a disjunction $ag\_spec = (ag\_id_1 \vee \cdots \vee ag\_id_n)$. We can then say that, if $ag\_spec \in \mathcal{AS}_{\mathcal{AGI}}$ and $ag\_spec' = ex'_{\vee}$, *ag_spec* is considered more specific than *ag_spec'* iff for each agent ag_i denoted by *ag_spec*, there exists at least a disjunct ex'_j in $ex'_{\vee}$ such that ag_i is included in the subset of agents denoted by ex'_j.

All these rules are enforced by the notion of *stronger agent specification*, which is formally defined as follows.

Definition 3.2 (Stronger Agent Specification) Let $ag\_spec, ag\_spec' \in \mathcal{AS}$ be two agent specifications applying to an agent *ag*. We say that *ag_spec* is stronger than *ag_spec'*, denoted $ag\_spec > ag\_spec'$, iff $ag\_spec \sqsubset ag\_spec'$. $\square$

Example 3.1 (Stronger Agent Specification) In this example, we consider a version of the subject class hierarchy depicted in Figure 3(b), extended with an additional class, namely SUBJ.ADMINISTRATIVE, direct child of class SUBJ.PERSON. Moreover, we suppose that a teacher can be also part of the administrative staff. Consider now the following agent specifications, all applying to the subject with identifier `Ann`, a tutor of 18 years old:

- $ag\_spec_1 = (\textsc{subj.person}(X).\texttt{age} > 16)$;
- $ag\_spec_2 = (\textsc{subj.student}(X))$;
- $ag\_spec_3 = (\textsc{subj.teacher}(X) \wedge \textsc{subj.administrative}(X) \vee \textsc{subj.tutor}(X))$;
- $ag\_spec_4 = \{\texttt{Ann}\}$.

We say that ag_spec_2, ag_spec_3, and ag_spec_4 are stronger than ag_spec_1 since, respectively:

- SUBJ.STUDENT is child of SUBJ.PERSON;
- SUBJ.TEACHER, SUBJ.ADMINISTRATIVE, and SUBJ.TUTOR are all children of SUBJ. PERSON;
- ag_spec_4 is an explicit agent specification.

Moreover, ag_spec_3 is stronger than ag_spec_2, since SUBJ.TUTOR is a child of SUBJ. STUDENT, whereas SUBJ.STUDENT is neither a child of SUBJ.TEACHER nor of SUBJ. ADMINISTRATIVE. Finally, ag_spec_4 is stronger than both ag_spec_2 and ag_spec_3 because it is an explicit agent specification.

If the condition expressed in Definition 3.2 is not satisfied, we say that agent specifications *ag_spec* and ag_spec' are *equally strong* (denoted $ag\_spec <> ag\_spec'$). Note that the "<>" symbol denotes agent specifications satisfying one of the following conditions:

- $ag\_spec = ag\_spec'$;
- $ag\_spec \neq ag\_spec'$, $ag\_spec \not> ag\_spec'$, and $ag\_spec \not< ag\_spec'$.

In such a case, the operations specified in the policies are taken into account, in order to determine the strongest with respect to the operation hierarchy. If this does not allow us to solve the conflict, we consider the policy with the *stronger* sign. As the operation hierarchy, the *stronger policy sign*, denoted by *str_sign*, depends on the application domain, and is stated when generating model instances (see Section 4). Finally, if policies have the same sign, the prevailing is the one with the strongest supervision mode.

We can now formally define the notion of stronger filtering policy. For this purpose, we use symbols ">" and "<>" to denote, respectively, the stronger and equally strong component of two conflicting policies.

Definition 3.3 (Stronger Filtering Policy) Consider the following two conflicting filtering policies, applying to a subject *subj* and an object *obj*:

- $fp = (sv\_id, (subj\_spec, obj\_spec), (op\_type, sign), sv\_mode)$
- $fp' = (sv\_id', (subj\_spec', obj\_spec'), (op\_type', sign'), sv\_mode')$

We say that fp is stronger than fp', denoted $fp > fp'$, iff one of the following conditions holds:

- $sv > sv'$;
- $sv <> sv'$ and $subj\_spec > subj\_spec'$;
- $sv <> sv'$, $subj\_spec <> subj\_spec'$, and $obj\_spec > obj\_spec'$;
- $sv <> sv'$, $subj\_spec <> subj\_spec'$, $obj\_spec <> obj\_spec'$, and $op\_type > op\_type'$;
- $sv <> sv'$, $subj\_spec <> subj\_spec'$, $obj\_spec <> obj\_spec'$, $op\_type <> op\_type'$, and $sign > sign'$;
- $sv <> sv'$, $subj\_spec <> subj\_spec'$, $obj\_spec <> obj\_spec'$, $op\_type <> op\_type'$, $sign = sign'$, and $sv\_mode > sv\_mode'$. $\Box$

Example 3.2 (Stronger Filtering Policy) Consider the following filtering policies, described in Example 2.4:

- $fp_1 = (\texttt{John}, (\text{SUBJ.PERSON}(X), \text{OBJ.SEX}(X)), (\texttt{allow}, -), \texttt{normal})$
- $fp_2 = (\texttt{John}, (\text{SUBJ.TEACHER}(X) \vee \text{SUBJ.TUTOR}(X), \text{OBJ.SEX}(X)), (\texttt{allow}, +), \texttt{normal})$
- $fp_3 = (\texttt{John}, (\text{SUBJ.STUDENT}(X).\texttt{age} > \texttt{14}, \text{OBJ.GYNECOLOGY}(X)), (\texttt{allow}, +), \texttt{normal})$
- $fp_4 = (\texttt{John}, (\text{SUBJ.PERSON}(X), \{\texttt{www.example.org}\}, (\texttt{allow}, +), \texttt{normal})$
- $fp_5 = (\texttt{Ted}, (\text{SUBJ.STUDENT}(X).\texttt{age} > \texttt{14}, \text{OBJ.GYNECOLOGY}(X)), (\texttt{allow}, +), \texttt{strict})$
- $fp_6 = (\texttt{Jane}, (\{\texttt{Bob}\}, \text{OBJ.GYNECOLOGY}(X)), (\texttt{notify}, -), \texttt{normal})$

It is easy to verify that policies fp_2, fp_3, and fp_4 are in conflict with fp_1. Nonetheless, according to our conflict resolution mechanism, fp_2 is more specific than fp_1, since TEACHER and TUTOR are children of PERSON, whereas fp_3 is more specific than fp_1, since STUDENT is a child of PERSON, and GYNECOLOGY is a child of SEX. Finally, fp_4 is more specific than fp_1, since the object specification is explicit. As a consequence, fp_2, fp_3, and fp_4 prevail over fp_1.

We now consider the policies specified by Jane and Ted. It is clear that policy fp_5 is in conflict with fp_3 wrt the supervision mode: nonetheless, the former is stronger, since the authority of Ted is stronger than John's one (i.e., agents associated with the supervisor class TEACHER have more authority than those associated with the supervisor class ADMINISTRATOR). Note that if John and Ted had the same authority, fp_5 would still prevail over fp_3, since it is associated with a `strict` supervision mode, which is stronger than the `normal` one in fp_3. Finally, policy fp_6 is in conflict with both fp_3 and fp_5, because of the operation and the sign. Nonetheless, since Jane, who is a parent, has more authority than John and Ted, fp_6 is the prevailing policy, even though the `notify` operation is weaker than `allow`, and the `normal` supervision mode is weaker than the `strict` one. However, if Jane had the same authority of John and Ted, fp_6 would still be the prevailing policy, since it uses an explicit subject specification.

3.2 Policy Validation and Optimization

To improve the efficiency of the filtering procedure, we make use of some precomputational strategies to control whether policies are 'valid'—i.e., whether the agent who specified a policy is really supervisor of the subjects to whom the policy applies—and to optimize them, by removing redundancy in the subject/object specifications. Thanks to these features, on one hand, we can grant that policies are specified only by authorized supervisors, and, on the other hand, we can reduce the computational costs at runtime when searching for the strongest policy applying to a given subject and a given object.

Policy validation is performed by comparing the subject specification in the policy with the set of supervision specifications stored in the *supervision specification base* $\mathcal{SVSB}$, and related to the agent who requests the insertion of the new policy. The principle is that, if the set of subjects to which the policy applies is included in or equal to the set of supervised subjects of the agent who specified such policy, the policy is accepted, otherwise it is refused. As an example, if the set of supervised users of an agent *sv* is denoted by $su\_spec_1, \ldots, su\_spec_n$, and *subj_spec* is the subject specification in the specified policy, such policy is considered valid iff $subj\_spec \sqsubseteq (su\_spec_1 \vee \cdots \vee su\_spec_n)$.

After validation, we verify whether the subject/object specifications in the policies can be expressed in a more compact form, since this speeds up the policy evaluation procedure at runtime. Redundacy may occur when the subject and/or object specifications in a policy consist of compound expressions which can be expressed in a simpler form without modifying their semantics. For instance, expression SUBJ.PERSON$(X) \wedge$ SUBJ.STUDENT(X) is redundant, since students are also persons (i.e., SUBJ.STUDENT is a child of SUBJ.PERSON), and thus it is equivalent to SUBJ.STUDENT(X). For the same reason, also SUBJ.PERSON$(X) \vee$ SUBJ.STUDENT(X) is redundant, and it is equivalent to SUBJ.PERSON(X). Similarly, SUBJ.PERSON(X).age $> 14 \wedge$ SUBJ.PERSON(X).age > 16 is equivalent to SUBJ.PERSON(X).age > 16, whereas SUBJ.PERSON(X).age $> 14 \vee$ SUBJ.PERSON(X). age > 16 is equivalent to SUBJ.PERSON(X).age > 14. A similar reduction can be applied also in the case of predicates of different type. For instance, SUBJ.PERSON$(X) \wedge$ SUBJ.STUDENT(X).age > 14 can be reduced to SUBJ.STUDENT(X).age > 14.

The advantages of redundancy reduction in terms of efficiency are based on the fact that evaluating complex expressions is computationally more expensive than evaluating simpler ones. For instance, in order to evaluate expression SUBJ. PERSON$(X) \wedge$ SUBJ.STUDENT(X), we need first to identify the subjects associated with the class SUBJ.PERSON(X), then those associated with class SUBJ.STUDENT(X), and finally we must compute the intersection of the two sets. By contrast, the same result can be obtained by evaluating the equivalent expression SUBJ.STUDENT(X), where we need only to identify the subjects associated with class SUBJ.STUDENT, thus performing one operation instead of three.

In all the previous examples, redundancy reduction is performed by removing predicates from a compound expression. More precisely, in case of conjunctions, we remove the predicate which includes the other one, whereas, in case of disjunctions, we remove the predicate which is included by the other one. A different case concerns expressions where redundancy reduction entails also the modification of the existing predicates. For instance, $\textsc{subj.student}(X)$ $\wedge$ $\textsc{subj.person}(X).\texttt{age} > 14$ is equivalent to $\textsc{subj.student}(X).\texttt{age} > 14$, which is obtained by removing the former predicate, and by adding its attribute constraint to the latter. In such a case, $\textsc{subj.student}(X)$ is not included in $\textsc{subj.person}(X).\ \texttt{age} > 14$ (i.e., not all the students are more than 14 years old). Nonetheless, the expression is redundant, since the intersection of the sets of subjects denoted by the two conjuncts is not empty. It is important to note that this occurs only with conjunctions: in fact, the disjunction $\textsc{subj.person}(X).\texttt{age} > 14 \vee \textsc{subj.student}(X)$ is not redundant, and it cannot be expressed in a more compact form.

In order to formally describe our redundancy reduction mechanism, we denote the general form of a predicate $\mathit{pred} \in \mathit{Pred}$ by $\mathit{pred}_{\mathcal{C}}.c_{\mathcal{A}}$, where $\mathit{pred}_{\mathcal{C}}$ is the class constraint component (equivalent to a class predicate), whereas $c_{\mathcal{A}}$ is either an attribute constraint, in case $\mathit{pred} \in \mathit{Pred}_{\mathcal{A}}$, or it is empty, in case $\mathit{pred} \in \mathit{Pred}_{\mathcal{C}}$.

Redundancy reduction can then be achieved by iteratively modifying an expression according the following rules, until none of them applies:

Rule 1 If *ex* is a conjunction, for each pair of conjuncts $\mathit{pred}_{\mathcal{C}}.c_{\mathcal{A}}, \mathit{pred}'_{\mathcal{C}}.c'_{\mathcal{A}}$ in *ex* such that $\mathit{pred}_{\mathcal{C}}.c_{\mathcal{A}} \sqsubseteq \mathit{pred}'_{\mathcal{C}}.c'_{\mathcal{A}}$, $\mathit{pred}_{\mathcal{C}}.c_{\mathcal{A}} \wedge \mathit{pred}'_{\mathcal{C}}.c'_{\mathcal{A}} \rightarrow \mathit{pred}_{\mathcal{C}}.c_{\mathcal{A}}$;

Rule 2 If *ex* is a conjunction, for each pair of conjuncts $\mathit{pred}_{\mathcal{C}}.c_{\mathcal{A}}, \mathit{pred}'_{\mathcal{C}}.c'_{\mathcal{A}}$ in *ex* such that $\mathit{pred}_{\mathcal{C}} \sqsubseteq \mathit{pred}'_{\mathcal{C}}$, $\mathit{pred}_{\mathcal{C}}.c_{\mathcal{A}} \wedge \mathit{pred}'_{\mathcal{C}}.c'_{\mathcal{A}} \rightarrow \mathit{pred}_{\mathcal{C}}.c_{\mathcal{A}} \wedge \mathit{pred}_{\mathcal{C}}.c'_{\mathcal{A}}$;

Rule 3 If *ex* is a disjunction, for each pair of disjuncts $ex_{\wedge}, ex'_{\wedge}$ in *ex* such that $ex_{\wedge} \sqsubseteq ex'_{\wedge}$, $ex_{\wedge} \vee ex'_{\wedge} \rightarrow ex'_{\wedge}$.

Before applying these rules, an agent specification *ag_spec* is put in DNF. Then, each disjunct in *ag_spec* is reduced according to rules 1 and 2, respectively, until they apply. For example, $\textsc{subj.person}(X) \wedge \textsc{subj.student}(X).\texttt{age} > 16$ can be reduced in a single step by applying rule 1. By contrast, $\textsc{subj.student}(X)$ $\wedge$ $\textsc{subj.person}(X).\texttt{age} > 14$ can be reduced in two steps, by applying rule 2 and then rule 1 as follows:

Rule 2 $\textsc{subj.student}(X) \wedge \textsc{subj.person}(X).\texttt{age} > 14 \rightarrow \textsc{subj.student}(X)$ $\wedge$ $\textsc{subj.student}(X).\texttt{age} > 14$;

Rule 1 $\textsc{subj.student}(X) \wedge \textsc{subj.student}(X).\texttt{age} > 14 \rightarrow \textsc{subj.student}(X).$ $\texttt{age} > 14$.

After each disjunct has been normalized, *ag_spec* is finally reduced according to rule 3, until it applies. Thus, given an agent specification $\textsc{subj.person}(X)$ $\wedge$ $\textsc{subj.}\ \textsc{student}(X).\texttt{age} > 16 \vee \textsc{subj.student}(X) \wedge \textsc{subj.person}(X).\texttt{age}$ > 14, it will be transformed as follows:

- $(\text{SUBJ.PERSON}(X) \wedge \text{SUBJ.STUDENT}(X).\texttt{age} > 16) \vee (\text{SUBJ.STUDENT}(X) \wedge \text{SUBJ. PERSON}(X).\texttt{age} > 14)$;
- $(\text{SUBJ.STUDENT}(X).\texttt{age} > 16) \vee (\text{SUBJ.STUDENT}(X).\texttt{age} > 14)$;
- $\text{SUBJ.STUDENT}(X).\texttt{age} > 14$.

Redundancy reduction rules are applied also to normalize compound expressions in supervision specifications, which may have the same redundancy problems of the subject and object specifications in filtering policies.

3.3 Policy Evaluation

Once policies have been validated and optimized, they can be used for filtering purposes. Filtering deals with verifying whether an object *obj* is appropriate for the subject *subj* who requested it, according to the filtering policies in the *filtering policy base* $\mathcal{FPB}$. In MFM, filtering enforcement requires a filtering mechanism able to evaluate the access requests of the subjects in $\mathcal{S}$ according to the filtering policies in $\mathcal{FPB}$. An access request can be represented as a pair (*subj_id*, *obj_id*), where *subj_id* is the identifier of the subject *subj*, submitting the access request, and *obj_id* is the identifier of the requested object *obj*. The evaluation of an access request is performed according to a filtering procedure, the first step of which is to identify the subset of filtering policies in $\mathcal{FPB}$ which are associated with subject *subj*, specified by one of his/her supervisors, applying to object *obj*. This subset is denoted by the notion of *policy base projection*, which is the subset of the policy base relevant to decide the answer to an access request. In order to formally define such notion, we introduce the function $Ag : \mathcal{AS} \rightarrow \mathcal{AG}$, which returns the set of agents denoted by the agent specifications in $\mathcal{AS}$. Moreover, we denote by SVI_{subj_id} the set of identifiers of the supervisors of subject with identifier *subj_id*.

Definition 3.4 (Policy Base Projection) Let (*subj_id*, *obj_id*) be an access request, and let $\mathcal{FPB}$ be a filtering policy base. The projection of $\mathcal{FPB}$ wrt (*subj_id*, *obj_id*), denoted $\Pi_{(subj\_id, obj\_id)}(\mathcal{FPB})$, is the subset of $\mathcal{FPB}$ such that: $\forall fp \in \mathcal{FPB}$, $fp \in \Pi_{(subj\_id, obj\_id)}(\mathcal{FPB})$ iff $sv\_id(fp) \in SVI_{subj\_id}$, $subj \in Ag(subj\_spec(fp))$, and $obj \in Ag(obj\_spec(fp))$. □

An algorithm enforcing the filtering procedure is reported in Figure 4. Algorithm 3.1 receives as input an access request (*subj_id*, *obj_id*) and a filtering policy base $\mathcal{FPB}$. It first computes the policy base projection. If it is empty, or if it is not possible to identify the strongest policy—in other words, when an access request cannot be evaluated—, the system performs the predefined default action, which is stated when generating an MFM instance (see Section 4). Otherwise, the system performs the action determined by the strongest filtering policy.

The strongest policy is determined by the *FindStrongFp*() function, illustrated in Figure 5. Function *FindStrongFp*() receives as input the projection, and assigns the policies in *ActFp* to the array *EvalFp* (step 1). Then it evaluates

Algorithm 3.1 Filtering Algorithm

```
INPUT:    1) An access request $(subj\_id, obj\_id)$,
          2) The filtering policy base $\mathcal{FPB}$
OUTPUT:   1) $SysAction = StrongFpAct$, if $StrongFpAct \neq$ NULL
          2) $SysAction =$ DEFAULT ACTION, otherwise
METHOD:

  1. Compute $\Pi_{(subj\_id, obj\_id)}(\mathcal{FPB})$
  2. If $\Pi_{(subj\_id, obj\_id)}(\mathcal{FPB}) \neq \emptyset$
         $ActFp \leftarrow \Pi_{(subj\_id, obj\_id)}(\mathcal{FPB})$
         If $|ActFp| = 1$: $StrongFpAct \leftarrow act(ActFp)$
         else: $FindStrongFp(ActFp)$
         If $StrongFpAct \neq$ NULL: $SysAction \leftarrow StrongFpAct$
         else: $SysAction \leftarrow$ DEFAULT ACTION
     else: $SysAction \leftarrow$ DEFAULT ACTION
```

Figure 4: Filtering Algorithm

each component of the policies in order to identify the strongest one (steps 2-6), according to the rules in Definition 3.3. Step 2 reduces the set of policies in *EvalFp*, until the remaining are equally strong with respect to the supervisor authority. If the resulting set *EvalFp* still consists of more than one policy, the procedure is iterated by taking into account the subject specification (step 3), the object specification (step 4), the operation type (step 5), the sign (step 6), and the supervision mode (step 7). Finally (step 8), the resulting set of policies in *EvalFp* is considered: if it consists of only one policy, the corresponding action is assigned to the variable *StrongFpAct*; otherwise, two different scenarios are possible:

1. *All the policies in EvalFp specify the same action.* In such a case, this action is assigned to the variable *StrongFpAct*.
2. *The policies in EvalFp specify different actions.* This may happen when diverse but equally strong actions are supported. In such a case, it is not possible to determine the prevailing policy, and thus *StrongFpAct* is set to NULL.

4 MFM Implementations

MFM can be used in a given application domain by generating an *instance* of the model, which defines the domain characteristics with respect to the agents, the agent class hierarchies, the type of the supported operations and of policy signs. In the instance, the operation hierarchy and the stronger policy sign are also specified, along with the default action to be performed by the system when an access request cannot be evaluated, and the supported supervision modes.

An MFM instance is formally defined as follows.

```
Function $FindStrongFp(ActFp)$

1. $EvalFp \leftarrow ActFp$

2. While $\forall fp \in EvalFp\ \exists fp' \in EvalFp$ such that $sv(fp) > sv(fp')$:
   do
       For each $fp, fp' \in EvalFp$ such that $sv(fp) > sv(fp')$:
           $EvalFp \leftarrow EvalFp - \{fp'\}$
       endfor
   endwhile

3. While $|EvalFp| > 1 \wedge \forall fp \in EvalFp\ \exists fp' \in EvalFp$ such that $subj\_spec(fp) > subj\_spec(fp')$:
   do
       For each $fp, fp' \in EvalFp$ such that $subj\_spec(fp) > subj\_spec(fp')$:
           $EvalFp \leftarrow EvalFp - \{fp'\}$
       endfor
   endwhile

4. While $|EvalFp| > 1 \wedge \forall fp \in EvalFp\ \exists fp' \in EvalFp$ such that $obj\_spec(fp) > obj\_spec(fp')$:
   do
       For each $fp, fp' \in EvalFp$ such that $obj\_spec(fp) > obj\_spec(fp')$:
           $EvalFp \leftarrow EvalFp - \{fp'\}$
       endfor
   endwhile

5. While $|EvalFp| > 1 \wedge \forall fp \in EvalFp\ \exists fp' \in EvalFp$ such that $op\_type(fp) > op\_type(fp')$:
   do
       For each $fp, fp' \in EvalFp$ such that $op\_type(fp) > op\_type(fp')$:
           $EvalFp \leftarrow EvalFp - \{fp'\}$
       endfor
   endwhile

6. While $|EvalFp| > 1 \wedge \forall fp \in EvalFp\ \exists fp' \in EvalFp$ such that $sign(fp) > sign(fp')$:
   do
       For each $fp, fp' \in EvalFp$ such that $sign(fp) > sign(fp')$:
           $EvalFp \leftarrow EvalFp - \{fp'\}$
       endfor
   endwhile

7. While $|EvalFp| > 1 \wedge \forall fp \in EvalFp\ \exists fp' \in EvalFp$ such that $sv\_mode(fp) > sv\_mode(fp')$:
   do
       For each $fp, fp' \in EvalFp$ such that $sv\_mode(fp) > sv\_mode(fp')$:
           $EvalFp \leftarrow EvalFp - \{fp'\}$
       endfor
   endwhile

8. If $|EvalFp| = 1$: $StrongFpAct \leftarrow act(EvalFp)$
   else:
       If $\forall fp \in EvalFp\ \nexists fp' \in EvalFp$ such that $act(fp) \neq act(fp')$:
           Choose randomly a policy $fp \in EvalFp$
           $StrongFpAct \leftarrow act(fp)$
       else $StrongFpAct \leftarrow$ NULL
   endif
   return $StrongFpAct$
```

Figure 5: Function *FindStrongFp*()

Definition 4.1 (MFM Instance) An MFM instance *MI* is a tuple

$$((\mathcal{SV}, \mathcal{S}, \mathcal{O}), (\mathcal{CH}_{\mathcal{SV}}, \mathcal{CH}_{\mathcal{S}}, \mathcal{CH}_{\mathcal{O}}), (\mathcal{OPT}, \prec), (signs, str\_sign), def\_act, sv\_modes)$$

where:

- $\mathcal{SV}, \mathcal{S}, \mathcal{O} \in 2^{\mathcal{AG}}$ are, respectively, the sets of supervisors, subjects, and objects;
- $\mathcal{CH}_{\mathcal{SV}}, \mathcal{CH}_{\mathcal{S}}, \mathcal{CH}_{\mathcal{O}}$ are the class hierarchy merging all the supported metadata vocabularies used for describing the characteristics of, respectively, supervisors, subjects, and objects;
- $(\mathcal{OPT}, \prec)$ is the hierarchy of the supported operations;
- $signs \subseteq \{+, -\}$ is the policy sign set, whereas $str\_sign \in signs$ is the stronger policy sign;
- $def\_act = (op\_type, sign)$, which denotes the system default action;
- $sv\_modes \subseteq \{\texttt{strict}, \texttt{normal}, \texttt{light}\}$ is the set of supported supervision modes. □

In the following we present two examples of MFM instances, corresponding to how MFM has been applied in the context of the EU projects EUFORBIA and QUATRO, described in the following sections.

4.1 Web Users' Protection: The EUFORBIA Project

MFM was formerly implemented in the framework of EUFORBIA, a EU project focused on user protection from inappropriate Web content.[2] The aim of EUFORBIA was to address the drawbacks of existing rating and filtering approaches by

- supporting content labels based on metadata vocabularies (also referred to as *rating systems*) which allow one to provide an accurate and as far as possible objective description of Web resources;
- supporting policies taking into account users' characteristics, and not only their identity.

In the EUFORBIA project, supervisors and subjects correspond to Web users, whereas object corresponds to Web resources. Following, we denote the set of users and resources as $\mathcal{U}$ and $\mathcal{R}$, respectively, whereas the set of supervisors is $\mathcal{SV} \subseteq \mathcal{U}$. The supervisor and subject class hierarchies are similar to those depicted in Figure 3, but they may vary depending on the context, whereas the object class hierarchy is similar to the one depicted in Figure 2.

[2] Detailed information about EUFORBIA are available at the project Web site: `http://e-msha.msh-paris.fr/Agora/Tableaux%20de%20bord/Euforbia`.

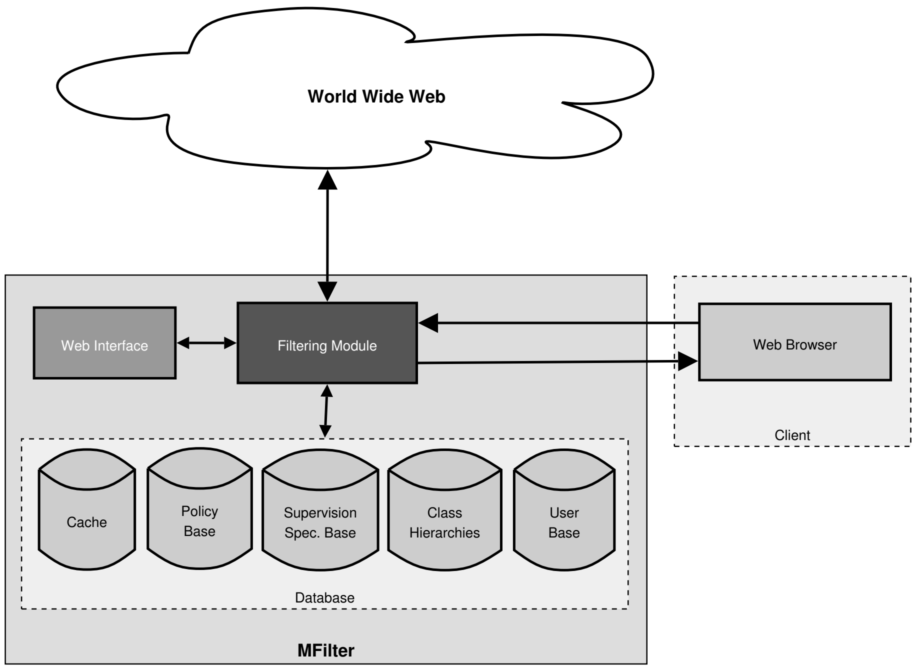


Figure 6: MFILTER Architecture

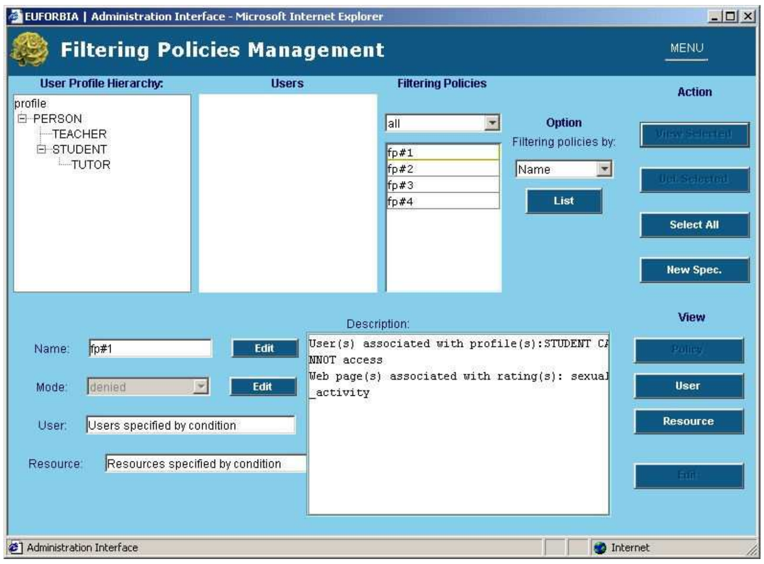
The EUFORBIA MFM instance supports only one operation type, which can be expressed as `allow_access` (`allow`, for short), and both the negative and positive signs, which determine whether a user can ("+") or cannot ("−") access a given resource. The EUFORBIA application domain has quite restrictive filtering requirements, since access to inappropriate resources must be absolutely prevented, even though this means that users cannot access possibly appropriate resources. Consequently, the stronger policy sign is the negative one, and, similarly, the action to be performed by the system in case an access request cannot be evaluated is ($\texttt{allow}, -$). Finally, all the available MFM supervision modes are supported.

The EUFORBIA MFM instance is then denoted as follows:

$$((\mathcal{SV}, \mathcal{U}, \mathcal{R}), (\mathcal{CH}_{\mathcal{SV}}, \mathcal{CH}_{\mathcal{U}}, \mathcal{CH}_{\mathcal{R}}), (\texttt{allow}), (\{+, -\}, -), (\texttt{allow}, -), \{\texttt{strict}, \texttt{normal}, \texttt{light}\})$$

MFM has been implemented in one of the two EUFORBIA filtering prototypes, called MFILTER, which addressed the needs of institutional users. After the end of the project, the prototype has been extended in order to support further functionalities, in particular the enforcement of supervised filtering. The main versions of the EUFORBIA MFM-based prototype are described in [6, 7].

Figure 6 depicts the architecture of the implemented prototype. MFILTER



Filtering Policies: all

fp#1
fp#2
fp#3
fp#4

Filtering policies by: Name

Name: fp#1 Edit

Mode: denied Edit

User: Users specified by condition

Resource: Resources specified by condition

Description:

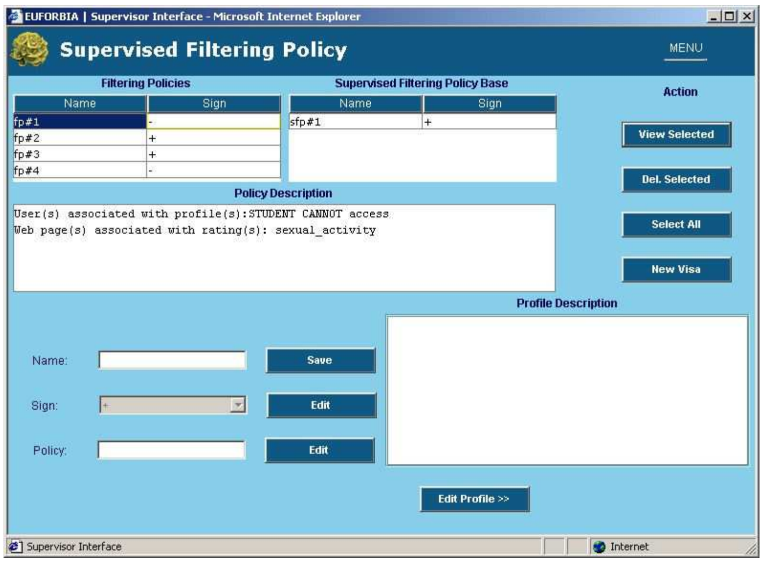
```
User(s) associated with profile(s):STUDENT CA
NNOT access
Web page(s) associated with rating(s): sexual
_activity
```

Figure 7: Filtering Policies Management main page

is a Java-based system, built on top of the Oracle DBMS, and it is structured into three main components. The first is the Filtering Module, which intercepts each access request submitted by users, and verifies whether it can be granted or not according to the filtering policies specified by the System Administrator (SA) and stored in the MFilter Database. The second is the Database, which stores all the information needed by the system to perform the filtering tasks. The third is the Web Interface, structured into three main components (the *Administration*, the *Supervision*, and the *User Interfaces*), which allows the management of the system, user authentication, and the specification/validation of policies.

Finally, in order to simplify the task of supervisors, in MFilter filtering policies are specified only by the SA, by using the Filtering Policy Management section of the Administration Interface (see Figure 7), whereas supervisors can only validate them. More precisely, they can display all the policies concerning their own supervised users, and decide whether they are valid or not. Whenever a policy is in/validated, the system generates a policy equivalent to the one specified by the SA, which has the identifier of the supervisor and the same sign, if the policy is valid, or a different sign, in case it is invalid. Moreover, if the subject specification in the administrator policies denotes a set of subjects which is a superset of the set of supervised users of the supervisor, in the supervisor policy it is substituted accordingly (see Example 4.1). The validation procedure is performed by using the Supervised Filtering Policy page, depicted in Figure 8.

Example 4.1 (Policy Specification in MFilter**)** Starting from the agents in Ex-



Filtering Policies

Name	Sign
fp#1	-
fp#2	+
fp#3	+
fp#4	-

Supervised Filtering Policy Base

Name	Sign
sfp#1	+

Policy Description

```
User(s) associated with profile(s):STUDENT CANNOT access
Web page(s) associated with rating(s): sexual_activity
```

Profile Description

Name:	
Sign:	+
Policy:	

Figure 8: The Supervised Filtering Policy Page

ample 2.1 and the supervision specifications in Example 2.3, consider the following policy, specified by administrator John:

$$fp_1 = (\texttt{John}, (\textsc{subj.person}(X), \textsc{obj.gynecology}(X)), (\texttt{allow}, +), \texttt{normal})$$

Ted, a teacher, and thus a supervisor for all the students, does not agree with it, and thus he states that it is invalid. Consequently, the system will generate the following policy:

$$fp_2 = (\texttt{Ted}, (\textsc{subj.student}(X), \textsc{obj.gynecology}(X)), (\texttt{allow}, -), \texttt{normal})$$

which is equivalent to fp_1 but has the identifier of Ted, a different sign (since it is invalid), and a subject specification denoting the subset of the subjects expressed by $\textsc{subj.person}(X)$, who are supervised by Ted.

4.2 Web Quality Assurance: The QUATRO Project

The application domain of the QUATRO project is more general than the one considered in EUFORBIA. QUATRO aims at defining a unified platform for ‘quality’ labels and trustmarks to be associated with Web resources. Quality labels do not describe the ‘quality’ of a resource, but rather they should be used in order to establish trust between content/service providers and end users, by advising the latter about the characteristics of the resource they are accessing. Such characteristics may concern the content of a resource, the authoritativeness and/or reliability of the information it provides, the privacy policies of a Web service, and so on. Consequently, the QUATRO platform is designed for any labelling vocabulary and for any application of Web content filtering, of which users' protection (as in EUFORBIA) is only a particular case. Finally, QUATRO addresses the need of enforcing strategies for granting labels' trustworthiness, an issue which has been neglected in the available rating and filtering approaches.[3]

The QUATRO MFM instance is similar to the EUFORBIA one with respect to supervisors, subject, and objects. The object class hierarchy consists currently of four metadata vocabularies, namely those of ICRA, IQUA, Segala, and WMA (see below), whereas the supervisor and subject class hierarchies depend on the context.

Differently from EUFORBIA, in QUATRO users' protection is not concerned: rather, the system is in charge mainly of notifying end users whether a resource is appropriate or not. Nonetheless, the user is provided also the possibility to block inappropriate content, and, consequently, two operation types are supported, namely, `notify` and `allow`, of which the latter is the stronger. The negative and positive signs are both supported, and in case of the `notify` operation type, they determine whether the user is notified that the requested resource is appropriate ("+") or not ("−"). Finally, similarly to EUFORBIA, the negative sign is the stronger, whereas the default action is $(\texttt{notify}, -)$. Since the aim of QUATRO is to make users aware of the content/characteristics of Web resource, and not to protect them, only the `normal` supervision mode is supported.

The QUATRO MFM instance is then denoted as follows:

$$((\mathcal{SV}, \mathcal{U}, \mathcal{R}), (\mathcal{CH}_{\mathcal{SV}}, \mathcal{CH}_{\mathcal{U}}, \mathcal{CH}_{\mathcal{R}}), (\texttt{notify} \prec \texttt{allow}), (\{+, -\}, -), (\texttt{notify}, -), \{\texttt{normal}\})$$

The QUATRO platform consists of two main components:

- an RDF schema for quality labels, which can be adopted by labelling services in order to provide a standard representation of their vocabularies and of the corresponding labels;
- a set of software tools, which collaborate in order to evaluate labels and to return the results of such evaluation to end users.

The definition of an RDF schema for labels aims at overcoming the drawbacks of the current scenario, where labelling authorities provide labels and trustmark which not only are stored in diverse formats, but, quite often, they are not even machine-understandable. Such standard format is the basis on which the QUATRO software tools are built. The first tool, the QUATRO proxy (referred to as QUAPRO), is in charge of evaluating the labels associated with Web resources, and to return the results to end users through the two other tools, the metadata visualizer (ViQ) and the search engine wrapper (LADI). An introduction to the QUATRO platform is provided by [11].

The current version of the RDF schema of QUATRO labels is described in [4], and it is the outcome of the collaboration of QUATRO with the W3C

[3] Detailed information about QUATRO are available at the project Web site: `http://www.quatro-project.org`.

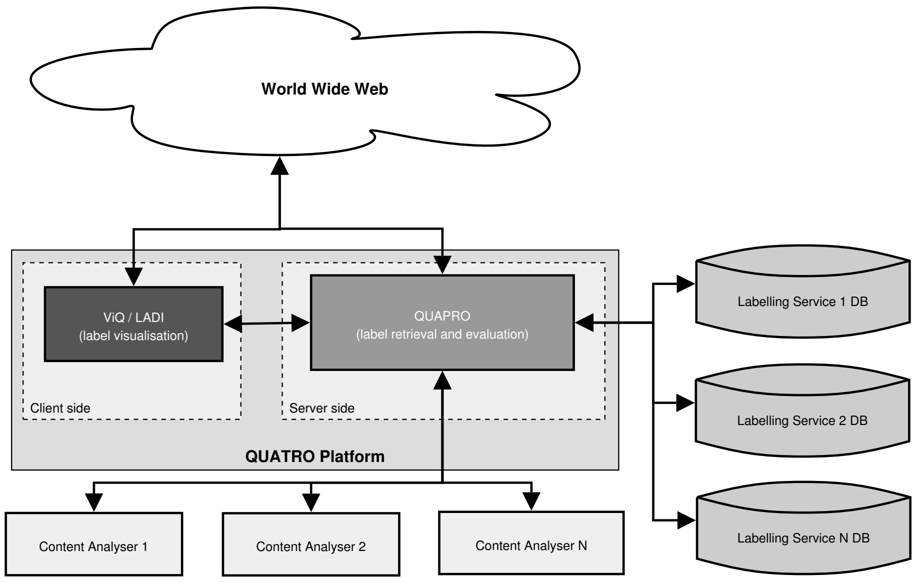


Figure 9: The QUATRO architecture

Semantic Web team and organizations interested in the diffusion of quality labels and trustmarks. The schema has been the basis for the definition of a general quality vocabulary, namely, the QUATRO vocabulary [2], and it has been adopted by the Internet rating associations partners of QUATRO (ICRA, WMA, and IQUA[4]), and by three certification agencies (PEGI Online, Segala, EIQUA[5]) external to the project. This effort has led to the establishment of a W3C Incubator Activity [3], currently under evaluation in order to be promoted as a W3C Recommendation Track.

The architecture of the QUATRO integrated system, depicted in Figure 9, consists of a Web service, QUAPRO, communicating with two front-end tools, ViQ and LADI, in charge of notifying users of the presence/absence of labels, and of displaying their content.

QUAPRO processes all the requests submitted by end-users through ViQ and LADI, in order to verify if labels are associated with the requested resource.

[4] ICRA (The Internet Content Rating Association: `http://www.icra.org`) provides labels for describing resources' content, and it has been already mentioned in the previous section. WMA (Web Mèdica Acreditada: `http://wma.comb.es`) and IQUA (The Internet Quality Agency: `http://www.iqua.net`) are trustmark agencies which certify the authoritativeness and reliability of Web sites providing services (in particular, WMA is specifically concerned with medical Web sites).

[5] PEGI Online (Pan-European Game Information Online: `http://www.pegi.info`) is a rating agency for interactive games. Segala (`http://www.segalamtest.com`) focuses on Web accessibility, in particular concerning mobile devices. EIQUA (Excellence Ireland Quality Association: `http://www.eiqa.com`) releases a trustmark, the W-Mark, certifying the quality and security level of e-Commerce Web sites.

QUAPRO is in charge also of checking the validity of labels. For this purpose, QUAPRO queries the databases of the labelling service which released the label, in order to verify both its integrity and expiry date. Additionally, QUAPRO may ask to a content analyzer to verify whether the description provided by the label actually corresponds to the content/characteristics of the requested resource. Actually, integrity, expiry date, and label-resource comparison are the three types of validity controls supported by the QUATRO platform: which of them should be performed is decided by the labelling service which released the label. For further details about label validation in QUATRO, we refer the reader to [11].

LADI is a search engine wrapper for Google and Yahoo!, which asks QUAPRO to verify whether labels are associate with each search result, and it displays an icon, in case a label is present. By clicking it, end-users can display the content of the label, along with its validity status.[6]

By contrast, ViQ is built as a browser extension for Mozilla Firefox, which, similarly to LADI, notifies users of the presence/absence of labels, and it allows them to display their content. The interface components of ViQ are a toolbar, a statusbar icon, and an entry in the browser's main menu, which allow the end user to access all the functionalities of the application. As an example, two screenshots of ViQ are depicted in Figures 10 and 11, showing, respectively, the messages displayed to the end-user when a label is detected, and the window displaying the content of a label. For a detailed description of the application, we refer the reader to [11].

ViQ, which is our contribution to the QUATRO software platform, is being extended in order to implement the MFM filtering approach, so that users will be notified not only of labels' validity status, but also of their in/appropriateness with respect to their characteristics and/or preferences. Filtering will be enforced as in MFilter, but some differences are present, due to the application domain and the system architecture.

Since ViQ is a client application, designed for end users, it is necessary to simplify as much as possible the task of policy specification, in order to grant usability. For this purpose, two strategies will be applied:

- Policy templates will be provided, stating filtering rules which can be general or for users with given characteristics. Such templates will be imported and possibly modified by users, in order to tailor them to their preferences.

- Since multiple vocabularies are supported, policy templates should be specified for any of them. Nonetheless, this may result in obliging users to import multiple policy templates in order to apply the same filtering rule on the supported vocabularies. In order to overcome such drawback, policy templates will be also provided based on general vocabularies, as the QUATRO one [2].

[6]LADI is publicly accessible at: `http://www.quatro-project.org/search.aspx`.

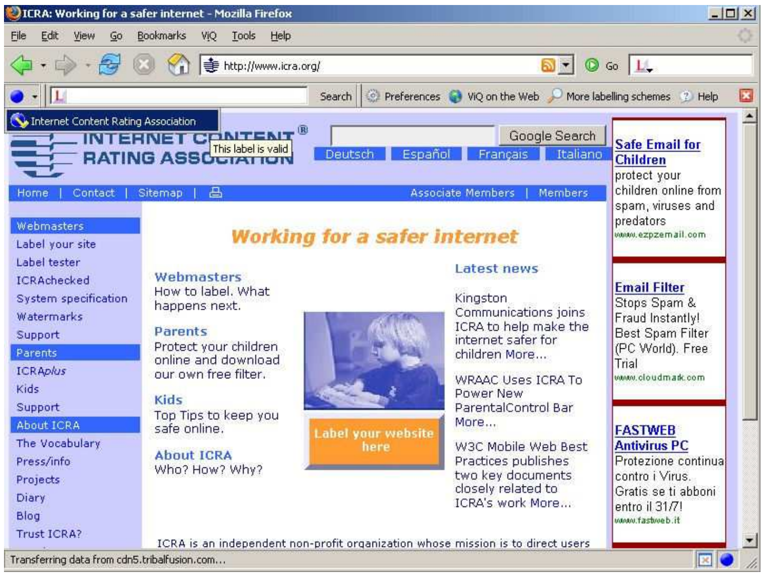
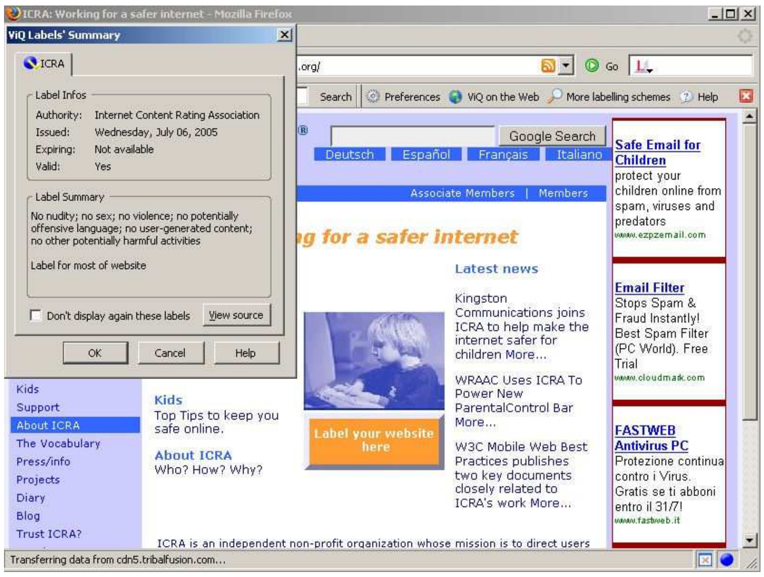

ICRA: Working for a safer internet - Mozilla Firefox
File Edit View Go Bookmarks ViQ Tools Help
http://www.icra.org/
Search Preferences ViQ on the Web More labelling schemes Help
Internet Content Rating Association
This label is valid
Working for a safer internet
Transferring data from cdn5.tribalfusion.com...

Figure 10: The available label and its validity status


ICRA: Working for a safer internet - Mozilla Firefox
ViQ Labels' Summary
ICRA
OK
Cancel
Help
View source
Transferring data from cdn5.tribalfusion.com...

Label Infos

Authority: Internet Content Rating Association
Issued: Wednesday, July 06, 2005
Expiring: Not available
Valid: Yes

Label Summary

No nudity; no sex; no violence; no potentially offensive language; no user-generated content; no other potentially harmful activities

Label for most of website

☐ Don't display again these labels

Figure 11: The window displaying labels' content

Users will be also in charge of specifying the supervisor and user profile hierarchies (i.e., the supervisor and subject class hierarchies). This may seem useless in a client context: we have a limited number of users, and it is more simple to apply specific filtering rules for each of them. Nonetheless, ViQ can be used also in situations where users' protection is concerned, and policy templates may be also considered as filtering profiles, determining the rules to be applied in case users have given characteristics (e.g., their age). Moreover, in a home context, parents would decide the type of resources which are in/appropriate for their children, and thus they perform a role of supervisor. Consequently, users will be provided the possibility to build profiles, starting from templates, which contain standard subject classes and attributes, used by policy templates.

5 Related Work

Web content filtering is currently provided by a number of Web applications as a service for parental control purposes, and for regulating the access to Web content of users connected to the networks of enterprises, libraries, schools, etc. Examples of the most diffused Internet filters are squidGuard (`www.squidguard.org`), DansGuardian (`dansguardian.org`), SonicWALL CFS (`www.sonicwall.com`), and the SurfControl Web Filter (`www.surfcontrol.com`). All such applications enforce the list-based approach, and some of them (such as DansGuardian) support also content analysis and/or PICS-based filtering. Nonetheless, they are not based on a Web content filtering model. Rather, they integrate and optimize established techniques in the fields of data mining, firewall blocking, and so on, in order to provide a service addressing the needs of very specific user categories, and granting a high degree of efficiency. If this is one of the most relevant features which make such applications usable, it has a major drawback of enforcing a quite restrictive filtering. In fact, none of them supports knowledge representation tools more sophisticated than the available PICS vocabularies. Moreover, users' characteristics are not taken into account in the specification of policies, since the same policies apply to all users.

By contrast, Web content filtering share several similarities with access control approaches, where policies are based on the subjects' and/or objects' characteristics. Actually, Web content filtering can be considered as an extension to access control, since it can be used both to protect objects from unauthorized subjects, and subjects from inappropriate objects. More precisely, the notion of agent specification in MFM has some similarities with the notions of *credential* and *role*.

Credentials have been originally proposed for access control in distributed environments [17], in order to denote authorized subjects by their characteristics, and not only by their identifiers, as in the traditional approach. Such notion has been furtherly developed and extended in [1], by formally defining of a constraint specification language and by organizing credentials into a hierarchy. Credentials are very similar to MFM agent classes, since they denote subjects by using a set of attribute-value pairs, and they may be hierarchically organized. The difference is that the notion of MFM agent class corresponds to the one of OWL class, and it is used to characterize not only subjects, but also objects and supervisors.

Roles-based access control [10, 15] addresses the same issues of credentials, but according to a different approach. In fact, roles do not store subjects' information: rather, they may be seen as access profiles to which a set of permissions is assigned. The actions which subjects are authorized to perform are then determined by associating with them one or more roles. The main difference between roles and the notion of agent class, as formalized in MFM, is that the latter is denoted by a set of attributes, and this allows us to specify policies directly for all the users whose attributes are associated with values satisfying a given condition (e.g., "all the users who are women and whose age is more than or equal to 18"). To obtain the same results by using roles, we need to specify a distinct role for each condition we would like to enforce (e.g., in order to specify the previous policy, we need to create a role corresponding to female users with the specified age).

However, besides this similarities, MFM has some distinguished features, like supervised filtering and the possibility of using different vocabularies to qualify both subjects and objects, which are not present in any credential- and role-based access control model. In particular, in [6] we have proposed a first model, focused on Web users' protection, which adopted the notion of credential to qualify subjects, whereas objects were characterized by using specific vocabularies. Starting from such work, in this paper we define a meta-model for Web content filtering, where a uniform representation of subjects and objects is given. Additionally, the supervised filtering component has been introduced, and support has been provided for different types of operations, hierarchically organized. Thanks to these features, MFM can be instantiated in order to be tailored to all the different domains into which Web content filtering can be applied.

Finally, MFM is compliant with the *rule-based* approach proposed by T. Berners-Lee et al. in [16] to enforce access control on the Web, by using RDF/OWL-based technologies. In [16], rules are Horn-like clauses, where each atom is an RDF triple, stating the set of requirements to be satisfied by a subject in order to access a given object. As mentioned above, MFM agent classes can be represented by OWL classes. Moreover, filtering policies can be expressed as Horn-like clauses, where the s-o specification component corresponds to the antecedent, whereas the action and supervision mode correspond to the consequent. The set of policies applying to a given agent can then be identified by unifying the class instances associated with an agent with the policies in $\mathcal{FPB}$. The conflict resolution mechanism described in Section 3.1 can be similarly expressed by a set of Horn-like clauses, which can be used to identify the strongest among a set of conflicting policies. The main difference between MFM policies and the access rules described in [16] is the evaluation of policies based not only on the subjects and objects to which they apply, but also on the supervisors who specified them. As such, MFM can be considered an extension to the rule-based approach proposed in [16].

6 Conclusions & Future Work

In this paper, extensions to Web content filtering have been proposed, which aim to overcome the current drawbacks. In particular, a *multi-strategy* approach has been described, which integrates the available techniques and focuses on the use of metadata for rating and filtering Web information. Such an approach consists of a filtering model, referred to as MFM, which provides a general representation of the Web content filtering domain, independently from its possible applications, and of two prototype implementations, partially carried out in the framework of the EU projects EUFORBIA and QUATRO, and designed for different application domains: users' protection and Web quality assurance, respectively.

Besides modelling the characteristics of the filtering domain, MFM improves the approaches currently available by supporting policies taking into account both users' and resources' characteristics, described by using multiple metadata vocabularies. As a result, MFM, on one hand, allows the enforcement of interoperability among filtering systems based on different approaches, and, on the other hand, it can be easily tailored to users' requirements and preferences.

MFM is also an attempt to define a framework compliant with the upcoming Web technologies. In particular, quite relevant are the recent efforts of the W3C in defining a standard architecture for Web services [8], which will incorporate Web content filtering into a broader framework, concerning the evaluation of online information with respect to given user requirements.

The complete integration of MFM into the Semantic Web and Web services framework is one of the main issue that will be addressed by future work. MFM is already compliant with RDF/OWL, but we plan to investigate how WSDL [9] and SOAP [13] can be used to express filtering policies. Provisional results of such research work will be already available in the final tools developed in the framework of the QUATRO project.

However, the major extension of MFM will concern the definition of strategies to be used in order to evaluate metadata trustworthiness, starting from the approach adopted in the QUATRO project. One of the main features to be supported is that the criteria to be used for validating metadata will be determined by evaluating both the statements possibly present in the metadata vocabularies, and the preferences expressed by users. Such extended version of MFM will be implemented in an enhanced and standalone version of the QUATRO metadata visualizer, which will perform independently all the tasks concerning metadata validation and policy evaluation.

References

[1] N. R. Adam, V. Atluri, E. Bertino, and E. Ferrari. A content-based authorization model for digital libraries. *IEEE Transactions on Knowledge and Data Engineering*, 14(2):296–315, March/April 2002.

[2] P. Archer. QUATRO vocabulary – Version 1.0. QUATRO Technical Specification, Apr. 2006. Available at: `http://www.quatro-project.org/vocabulary/1.0`.

[3] P. Archer. W3C Content Label Incubator Group. W3C Incubator Activity Web Site, World Wide Web Consortium, July 2006. `http://www.w3.org/2005/Incubator/wcl`.

[4] P. Archer, N. Shimuzu, K. Ahmed, D. Brickley, D. Appelquist, and K. Chandrinos. RDF content labels: Schema description. QUATRO Technical Specification, July 2005. Available at: `http://www.w3.org/2004/12/q/doc/content-labels-schema.htm`.

[5] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook. Theory, Implementation, and Applications.* Cambridge University Press, Cambridge, 2002.

[6] E. Bertino, E. Ferrari, and A. Perego. Content-based filtering of Web documents: The Ma$\mathcal{X}$ system and the EUFORBIA project. *International Journal of Information Security*, 2(1):45–58, Nov. 2003.

[7] E. Bertino, E. Ferrari, and A. Perego. Web content filtering. In E. Ferrari and B. Thuraisingham, editors, *Web and Information Security*, chapter 6, pages 112–132. IDEA Group Publishing, Hershey, PA, 2006.

[8] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web Services Architecture. W3C Working Group Note, Feb. 2004. Available at: `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211`.

[9] D. Booth and C. K. Liu. Web Services Description Language (WSDL) Version 2.0 Part 0: Primer. W3C Candidate Recommendation, Mar. 2006. Available at: `http://www.w3.org/TR/2006/CR-wsdl20-primer-20060327`.

[10] D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli, editors. *Role-Based Access Control.* Artech House Publishers, Norwood MA, 2003.

[11] V. Karkaletsis, A. Perego, P. Archer, K. Stamatakis, P. Nasikas, and D. Rose. Quality labeling of Web content: The QUATRO approach. In *Proc. of the WWW'06 Workshop on Models of Trust for the Web (MTW 2006)*, 2006. Available at: `http://www.l3s.de/~olmedilla/events/MTW06_papers/paper11.pdf`.

[12] C. Lagoze and J. Hunter. The ABC ontology and model. *Journal of Digital Information*, 2(2), Nov. 2001. Available at: `http://jodi.ecs.soton.ac.uk/Articles/v02/i02/Lagoze`.

[13] N. Mitra. SOAP – Version 1.2 Part 0: Primer. W3C Recommendation, June 2003. Available at: `http://www.w3.org/TR/2003/REC-soap12-part0-20030624`.

[14] P. Resnick and J. Miller. PICS: Internet access controls without censorship. *Communications of the ACM*, 39(10):87–93, Oct. 1996.

[15] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, Feb. 1996.

[16] D. J. Weitzner, J. Hendler, T. Berners-Lee, and D. Connolly. Creating a policy-aware Web: Discretionary, rule-based access for the World Wide Web. In E. Ferrari and B. Thuraisingham, editors, *Web & Information Security*, chapter 1, pages 1–31. IDEA Group Publishing, Hershey, PA, 2006.

[17] M. Winslett, N. Ching, V. Jones, and I. Slepchin. Using digital credentials on the World Wide Web. *Journal of Computer Security*, 5(3):255–266, 1997.