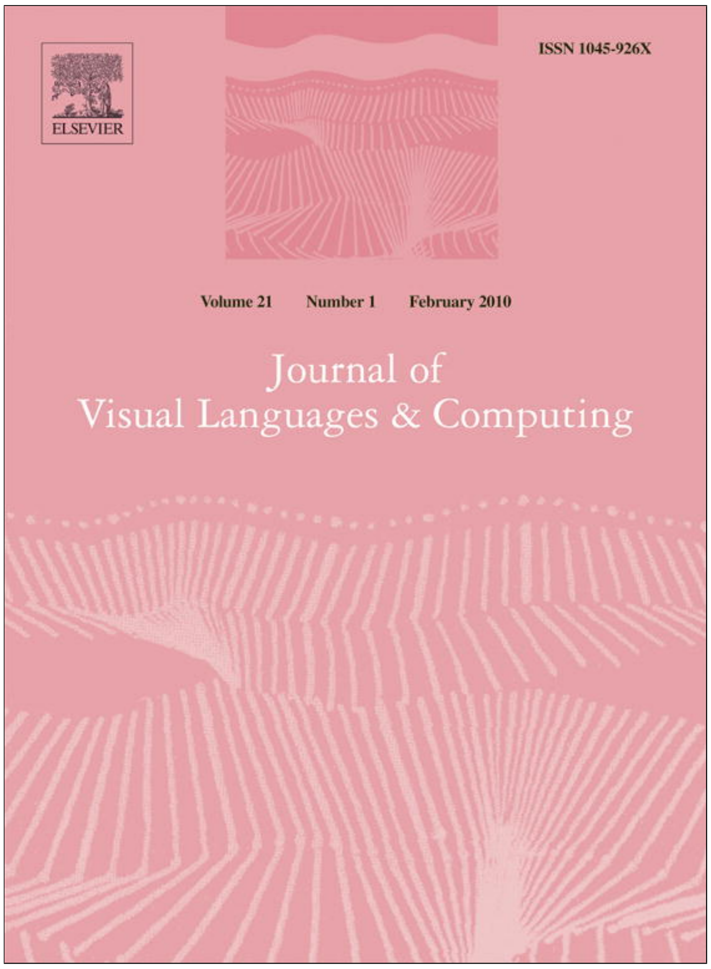

ISSN 1045-926X
ELSEVIER
Volume 21 Number 1 February 2010
Journal of Visual Languages & Computing

Journal of Visual Languages and Computing 21 (2010) 41–64

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

A system for visual role-based policy modelling

Massimiliano Giordano [a], Giuseppe Polese [a], Giuseppe Scanniello [b,*], Genoveffa Tortora [a]

[a] Dipartimento di Matematica e Informatica, University of Salerno, Via Ponte Don Melillo, 84084 Fisciano (SA), Italy
[b] Dipartimento di Matematica e Informatica, University of Basilicata, Viale Dell'Ateneo 10, Macchia Romana, 85100 Potenza, Italy

ARTICLE INFO

Article history:
Received 29 December 2008
Received in revised form
25 September 2009
Accepted 20 November 2009

Keywords:
RBAC
Role-based security policy
Visual languages
XACML
Eclipse IDE

ABSTRACT

The definition of security policies in information systems and programming applications is often accomplished through traditional low level languages that are difficult to use. This is a remarkable drawback if we consider that security policies are often specified and maintained by top level enterprise managers who would probably prefer to use simplified, metaphor oriented policy management tools.

To support all the different kinds of users we propose a suite of visual languages to specify access and security policies according to the role based access control (RBAC) model. Moreover, a system implementing the proposed visual languages is proposed. The system provides a set of tools to enable a user to visually edit security policies and to successively translate them into (eXtensible Access Control Markup Language) code, which can be managed by a Policy Based Management System supporting such policy language.

The system and the visual approach have been assessed by means of usability studies and of several case studies. The one presented in this paper regards the configuration of access policies for a multimedia content management platform providing video streaming services also accessible through mobile devices.

1. Introduction

Large and modern organizations need to handle a huge number of access rules and constraints for guaranteeing secure access to sensitive data within their information systems. They need security paradigms that can be easily understood and managed by enterprise managers, because they are the ones who have the knowledge about the enterprise and the way its resources should be accessed. To this end, different access models have been proposed, such as discretionary access controls [1], mandatory access controls [2], access control list [3], task based authorization [4], and the role-based [5,6]. They require the enterprise manager to have specific knowledge in order to be able to define access control policies based on request/response scenarios.

In the role-based access control (RBAC) model the policies relate information on users, resources, applications, security characteristics, factory priorities, and network features [7]. They describe the kind of user that can benefit from a given application, the priority in using it, and finally the network resources allocatable for that application. For example, priorities in using bandwidth might have to be defined for a company, whose employees need to access internet for visiting a Web site or for a crucial activity in a workflow. RBAC is particularly suitable to model policies in which the privileges to use a resource are connected to a role rather than to a specific user. Each user can belong to more than one role, and for each of them several privileges can be defined. When an employee changes its position in the organization chart, the manager assigns him/her a new role discarding the old one.

* Corresponding author.
E-mail addresses: mgiordano@unisa.it (M. Giordano), gpolese@unisa.it (G. Polese), giuseppe.scanniello@unibas.it (G. Scanniello). gtortora@unisa.it (G. Tortora).

1045-926X/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jvlc.2009.11.002

RBAC policies can be defined by using several access control languages. One of the most frequently used standardized language for defining policies is XACML (eXtensible Access Control Markup Language) [8], an XML based language to define actions and rules for subjects and targets. XACML defines two types of languages for modelling both control policies and request/response on resources. The first type of language is used to express access control policies, specifying who can make what, where, and when. The second type of language is used to define queries on whether a particular access should be allowed (requests), and to describe answers to such queries (responses), such as Permit, Deny, Indeterminate, and Not Applicable.

Although XACML provides a powerful abstraction for policy definition in heterogeneous frameworks, tools assisting administrators in the management of policies are highly desirable. Visual language based policy management systems should enable the high level specification and management of access policies. They introduce a further level of abstraction with respect to XACML, enabling the modelling of policies according to metaphors close to the specific application domain and to human reasoning. Thus, they can considerably simplify the work of enterprise security managers.

In this work we propose a visual language based system for specifying role based access policies and for implementing them in the XACML language. In particular, we describe a suite of visual languages enabling the management of role-based access policies [5,6] in heterogeneous frameworks and configurable networks, providing a metaphor oriented layer above the RBAC model. Moreover, the approach can be proficiently embedded within software engineering methodologies to specify access policies to be enforced during the design of information systems and applications.

The main visual language in our proposal is the Role Diagram, which is used to specify roles. It allows us to model roles and relations among them. Access policies are defined through the Permission Diagram, which allows an administrator to specify who can use the available resources. Constraints are described through the Separation of Duties Diagram. It is used to specify resources that cannot be employed by users who have played a given role, which must have been previously defined in the Role Diagram. Finally, in order to assign users to roles we propose the Role Assignment Diagram.

This set of visual languages has been embedded within a system providing editors to compose visual sentences and compilers to generate access policies abiding by the XACML standard. The system also includes a server for policy management and access request evaluation, and a development environment supporting policy design, which has been implemented as an *Ecplise* plug-in. The system has been used experimentally in several case studies. The one described in this paper regards the configuration of access policies for a multimedia content management platform providing video streaming services accessible through several types of devices, including mobile devices. We have previously used the system in the context of collaborative environments, and in Voice Over Ip infrastructures. We are currently using it in the context of domotics applications. A controlled experiment to compare our tool and the one of the most pertinent competitor, namely XGrid Tool, has been also conducted and the results have been presented and discussed as well.

The remainder of the paper is organized as follows. Section 2 briefly presents some concepts concerning visual languages, RBAC, and XACML. The proposed visual languages are detailed in Section 3, whereas the system prototype is illustrated in Section 4. The case study and system usability issues are discussed in Sections 5 and 6, respectively. In Section 7 we present the comparative evaluation between our tool and XGrid Tool which also uses a visual based approach. Related works are discussed in Section 8. Finally, discussion is provided in Section 9.

2. Background

In this section we briefly recall some basic concepts underlying the proposed system. These include visual languages, the role based access control model, and the eXtensible Access Control Markup Language.

2.1. Visual languages

Visual and diagrammatic representations play a central role in several application domains [9,10], since they provide important tools for describing and reasoning. As visual languages have been applied to new application domains, such as spatial databases, education, software engineering, and so forth, many different types of visual notations and underlying grammar models have been devised. As for visual language grammars, many different formalisms have been proposed in the literature [9]. Since in the rest of the paper we will describe several visual languages and will show how to model them through one of such formalisms, namely the Extended Positional Grammars (XPGs), in the following we will review the basic concepts underlying XPGs.

XPGs represent a direct extension of context-free string grammars. In particular, the XPG formalism is based on an extension of LR (Left to Right) parsing named XpLR (eXtended Positional Left to Right) methodology [11]. The XPG formalism conceives a sentence as a set of symbols with attributes. Such attributes can be classified in physic, syntactic, and semantic attributes. The physical component represents the features of a symbol and allows us to materialize the sentence to our senses; whereas the values of the syntactic attributes are determined by the relationships holding among the symbols. Thus, a sentence is specified by combining symbols with relations.

More formally, an Extended Positional Grammar is the pair (G, PE), where PE is a *positional evaluator*, and G is a particular type of context-free string attributed grammar $(N, T \cup POS, S, P)$ where:

- N is a finite non-empty set of *non-terminal* symbols;
- T is a finite non-empty set of *terminal* symbols, with $N \cap T = \emptyset$;

- *POS* is a finite set of *binary relation identifiers*, with POS $\bigcap N = \emptyset$ and $POS \cap T = \emptyset$;
- $S \in N$ denotes the starting symbol;
- *P* is a finite non-empty set of *productions* having the following format:

  $A \rightarrow x_1 R_1 x_2 R_2, \ldots, x_{m\text{-}1} R_{m\text{-}1} x_m \Delta, \Gamma$

  where *A* is a *non-terminal* symbol, each x_i is a symbol in $N \cup T$ and each R_j is partitioned in the following two sub-sequences:

  $(\langle REL_{j_1}^{h_1}(t_1), \ldots, REL_{j_k}^{h_k}(t_k)\rangle, \langle REL_{j_{k+1}}^{h_{k+1}}(t_{k+1}), \ldots, REL_{j_n}^{h_n}(t_n)\rangle)$

  with $1 \leq k \leq n$. Each $REL_{j_i}^{h_i}(t_i)$ relates physic or syntactic attributes of x_{j+1} with physic or syntactic attributes of $x_{j\text{-}h_i}$, where $0 \leq hi < j$, by means of a threshold t_i. The relation identifiers in the first sub-sequence of an R_j, named *driver relations*, are used during syntax analysis to determine the next symbol to be scanned; whereas the ones in the second sub-sequence, named *tester relations*, are used to check whether the last scanned symbol (terminal or non-terminal) is properly related to previously scanned symbols. We refer to the driver (tester, resp.) relations of R_j with *driver*(R_j) (*tester*(R_j), resp.). Δ is a set of rules used to synthesize the values of the syntactic/physic attributes of *A* from those of $x_1, x_2, \ldots, x_m$; Γ is a set of triples $(N_j, Cond_j, \Delta_j)_j = 1, \ldots, t, t \geq 0$, used to dynamically insert new symbols in the input visual sentence during the parsing process. In particular,
- N_j is a terminal symbol to be inserted in the input visual sentence;
- $Cond_j$ is a pre-condition to be verified in order to insert N_j;
- Δ_j is the rule used to compute the values of the syntactic attributes of N_j from those of $x_1, \ldots, x_m$.

Informally, a Positional Evaluator (PE) is a materialization function that transforms a linear representation into the corresponding graphical representation of the visual sentence. The *language described by an XPG, L(XPG)*, is the set of the visual sentences from the starting symbol *S* of XPG [11].

2.1.1. Example

In this example we introduce an XPG modelling state transition diagrams. Let $STD = (N, T \cup POS, S, P)$ be the XPG for State Transition Diagrams, characterized as follows. The set of non-terminals is given by $N = Graph, Node, Edge, NLabel, ELabel$, where the first two symbols have one attaching region as a syntactic attribute; *Edge* has two attaching points as syntactic attributes, whereas the last two symbols have two syntactic attributes, called head and tail, both specifying a position in the plane. *Graph* is the starting symbol.

The set of terminals is $T = NODEI$, *NODEIF*, *NODEF*, *NODEG*, *EDGE*, *a*, *b*, *DIGIT*, *PLACEHOLD*. The terminals *NODEI*, *NODEIF*, *NODEF*, *NODEG* represent the initial, the initial and final, the final, and the generic node, respectively, of a state transition diagram. As syntactic attributes they have one attaching region corresponding to the borderline of the node, and one containment area corresponding to the circle area representing the node. The terminal *EDGE* has two attaching points as syntactic attributes, corresponding to the start and end points of the edge. *PLACEHOLD* is a terminal to be dynamically inserted in the input sentence during the parsing process. It has one attaching region as syntactic attribute. The symbols *a* and *b* represent the labels of the edges and have two syntactic attributes, called head and tail, both specifying a position in the plane. Finally, *DIGIT* is a symbol whose visual pattern matches the decimal digits 0–9. It is used to compose node labels, and has two syntactic attributes, called head and tail, both specifying a position in the plane.

Typical instances of symbols for this language are graphically depicted in Fig. 1. Here, each attaching region is represented by a bold line and each is identified by a number; each containment area is represented by a light gray area, while the attaching points are represented by bullets.

The set of relations is given by $POS = LINK_{i,j}$, *any*, *contains*, *edge-labelling*, where

- $LINK_{i,j}$ is defined as follows: a symbol *x* is in relation $LINK_{i,j}$ with a symbol *y* iff attaching point (or region) *i* of *x* is connected to attaching point (or region) *j* of *y*, and will be denoted as *i_j* to simplify the notation. Moreover, we use the notation $\overline{ij}$ when describing the absence of a connection between two attaching areas *i* and *j*;
- the relation identifier *any* denotes a relation that is always satisfied between any pair of symbols;
- *contains* is a containment geometric relation. In particular, if A is a symbol with a containment area as syntactic attribute and *B* is a symbol, then *A contains B* if and only if *B* is inside the containment area of *A*.
- *edge-labelling* is a geometric relation. In particular, if *A* is a symbol of type EDGE and *B* is a symbol representing a string label, then *A edge-labelling B* if

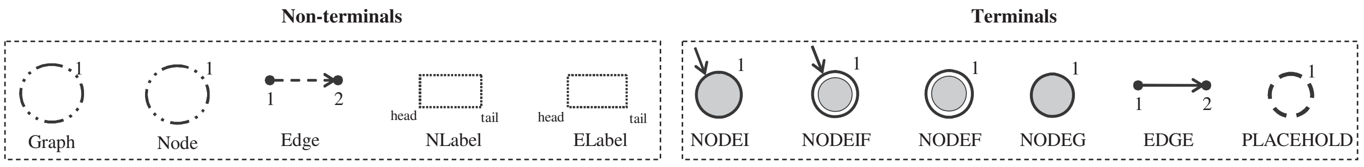


Fig. 1. Visual representation of non-terminals and terminals for the grammar STD.

and only if *B* is close to *A* with respect to their syntactic attributes.

Next, we provide the set of productions for describing State Transition Diagrams.

(1) $Graph \rightarrow NODEI \langle contains \rangle NLabel$
$\Delta : (Graph_1 = NODEI_1)$
(2) $Graph \rightarrow NODEIF \langle contains \rangle NLabel$
$\Delta : (Graph_1 = NODEIF_1)$
(3) $Graph \rightarrow Graph' \langle \langle 1\_1 \rangle, \langle \overline{1\_2} \rangle \rangle\ Edge2\_1\ Node$
$\Delta : (Graph_1 = Graph'_1\ \text{-}Edge_1)$
$\Gamma : (PLACEHOLD; |Node_1| > 1; PLACEHOLD_1 = Node_1\text{-}Edge_2)$
(4) $Graph \rightarrow Graph' \langle \langle 1\_1, 1\_2 \rangle \rangle Edge$
$\Delta : (Graph_1 = (Graph'_1\ \text{-}Edge_1)\text{-}Edge_2)$
(5) $Graph \rightarrow Graph' \langle \langle 1\_2, \overline{1\_1} \rangle \rangle\ Edge1\_1\ Node$
$\Delta : (Graph_1 = Graph'_1\ \text{-}Edge_2)$
$\Gamma : (PLACEHOLD; |Node_1| > 1; PLACEHOLD_1 = Node_1\text{-}Edge_1)$
(6) $Graph \rightarrow Graph' \langle \mathbf{any} \rangle PLACEHOLD$
$\Delta : (Graph_1 = Graph'_1\ + PLACEHOLD_1)$
(7) $Node \rightarrow NODEG \langle \mathbf{contains} \rangle NLabel$
$\Delta : (Node_1 = NODEG_1)$
(8) $Node \rightarrow NODEF \langle \mathbf{contains} \rangle NLabel$
$\Delta : (Node_1 = NODEF_1)$
(9) $Node \rightarrow PLACEHOLD$
$\Delta : (Node_1 = PLACEHOLD_1)$
(10) $Edge \rightarrow EDGE \langle \mathbf{edge\text{-}labelling} \rangle ELabel$
$\Delta : (Edge_1 = EDGE_1, Edge_2 = EDGE_2)$
(11) $NLabel \rightarrow DIGIT$
$\Delta : (NLabel_{head} = DIGIT_{head}, NLabel_{tail} = DIGIT_{tail})$
(12) $NLabel \rightarrow NLabel' \langle \mathbf{right\text{-}to}\ DIGIT$
$\Delta : (NLabel_{head} = NLabel'_{head}\ , NLabel_{tail} = DIGIT_{tail})$
(13) $ELabel \rightarrow a$
$\Delta : (ELabel_{head} = a_{head}, ELabel_{tail} = a_{tail})$
(14) $ELabel \rightarrow b$
$\Delta : (ELabel_{head} = b_{head}, ELabel_{tail} = b_{tail})$

Notice that $Graph_1 = Graph'_1$ indicates set difference and is to be interpreted as follows: “attaching area 1 of Graph has to be connected to whatever is attached to the attaching area 1 of *Graph*′ except for the attaching point 1 of EDGE”. Moreover the notation $Node_1$ indicates the number of connections to the attaching area 1 of Node.

According to these rules, a state transition diagram is described by a graph defined as

- an initial node containing a label (production 1) or as
- an initial–final node containing a label (production 2) or, recursively, as
- a graph connected to a node through an outgoing (production 3) or incoming (production 5) edge, or as
- a graph with a loop edge (production 4).

A node can be either a generic node containing a label (production 7) or a final node containing a label (production 8). An edge is labelled (production 10) by a (production 13) or b (production 14). A node label is the string concatenation of decimal digits (productions 11 and 12).

The reduction process of a typical State Transition Diagram is shown in Fig. 2. During the reduction process, the introduction of *PLACEHOLD* terminals (productions 3 and 5) and their successive processing (productions 6 and 9) allows us to keep knowledge of the source and the target node of each reduced edge. The same result could be achieved by using the terminal NODEG instead of *PLACEHOLD*. However this would let the grammar describe also unconnected graph structures.

2.2. Access control models

One of the most challenging problems in managing large and heterogeneous networks is the complexity of security administration. In particular, the definition and modelling of access control represents one of the open challenges. Access is meant as the use of a hardware or software resource, whereas access control is intended as the release of any resource or a restriction on its use. The Access Control List (ACL) model [3] has been recently used as a model for access control in large networked applications and permission systems. ACL restricts the access by verifying membership in static permission lists. A typical example of system using ACL is UNIX. Basically, UNIX assigns a user and a group to a process. The latter can manipulate a file if that user or group has permission to do so. There are two other basic types of access control mechanisms: discretionary access controls (DAC) [1], and mandatory access controls (MAC) [2], which are replaced by role-based access control (RBAC) [7]. In RBAC decisions are based on the roles that individual users have as part of an organization. The main advantages of RBAC are its flexibility and the reduced management overhead. Flexibility allows the administrator to enforce the principles of Least Privilege, conflict between duties, and dynamic and/or static separation of duties. Moreover, by exploiting the RBAC natural ability to describe the organizational structure of a distributed system we can also receive support in the choice of the administrative task to decentralize.

2.3. XACML

XACML (eXtensible Access Control Markup Language) [8] is an XML-based language, or schema, conceived to create access policies, automate their use in the management of access control for generic devices, and support interoperability among different systems and frameworks. It was designed to replace existing, application-specific, proprietary access-control mechanisms. Previously, every application vendor had to create its own customized method for specifying access control, and these typically could not talk to one another. The XACML specification describes both a request/response language for expressing queries about whether a particular access should be allowed, describing the answers to those queries, and an access control policy language, which allows developers to specify who can do what and when. In a typical XACML based scenario, a user willing to undertake some action on a particular resource submits a query to the entity protecting that resource, named Policy Enforcement Point (PEP). By using the XACML request language the PEP prepares a request based on user attributes such as action, resource, and other relevant

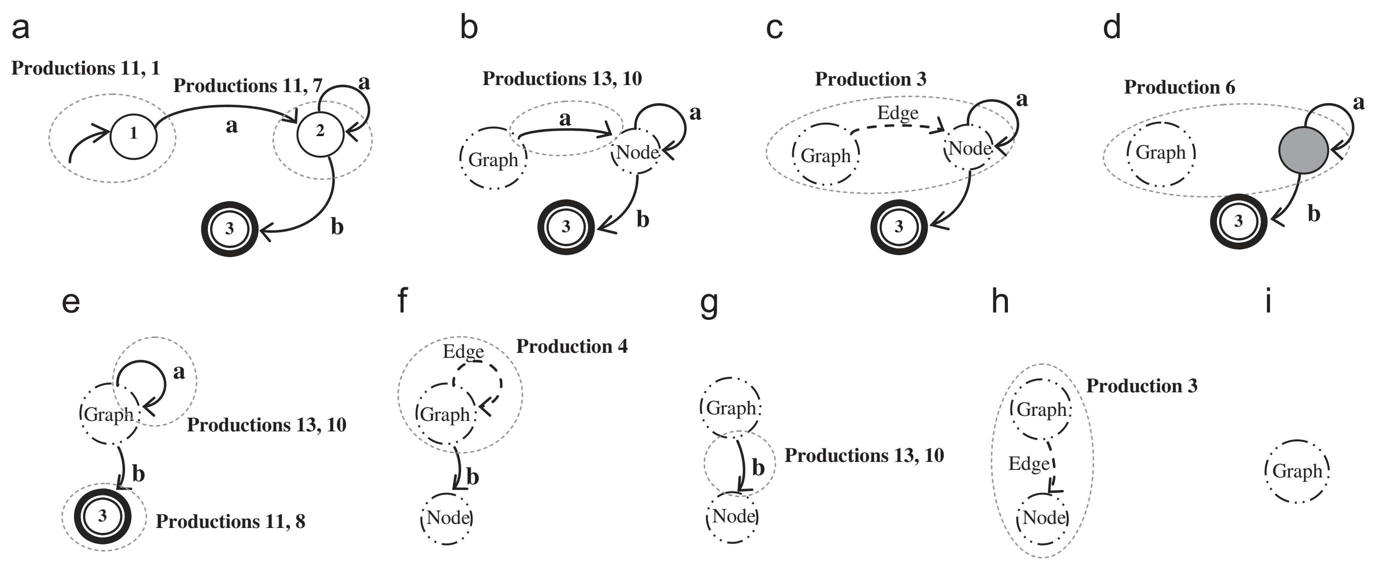


Fig. 2. The reduction process for a state transition diagram.

information. The request is sent to a Policy Decision Point (PDP), which examines the request, and retrieves policies written in the XACML policy language that are applicable to this request. The PDP also determines if the access can be granted according to the request.

That answer is returned to the PEP, which allows or denies user access. The answer returned to the PEP is expressed in the XACML response language.

3. Visual policy editor

As said above, the management of role-based XACML policies is a difficult task that cannot be accomplished by users without sufficient technical skills.

In this section we present a set of visual languages that can potentially simplify the work of an enterprise security manager by letting him/her directly manipulate role based security policies in a metaphor oriented fashion. Thus, the new usage scenario of role-based XACML security policies will be the one depicted in Fig. 3.

The security manager defines XACML policies by means of visual languages implemented in the Visual Policy Editor. Visually specified policies are translated into the XACML language by means of visual language compilers and stored on a permanent memory device. When the end user makes an attempt to access services and resources from a client, a request is generated. Then, a specific PEP intercepts and sends it to the PDP in the form of an authorization decision request. The PDP evaluates it against the stored policies, producing an authorization decision that is returned to the PEP responsible for the requested resource. In other words, the specific PEP is responsible for enforcing the decision to release the required resource or service. Thus, each service provider has to bind the service to the corresponding PEP.

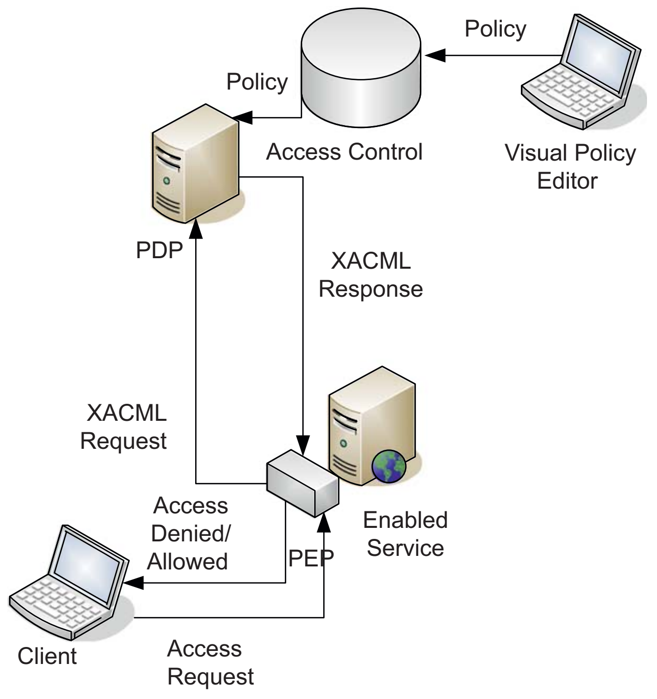


Fig. 3. Visual Role-based policy management scenario.

In this section we describe in details the suite of visual languages defined within the visual policy editor. The top-tier language is Role Diagram (RD), which allows us to define the roles and their hierarchic relations. Then, Permission Diagram (PD) allows us to define access permissions for a resource and to associate permissions to a role. The restriction Separation of Duties Diagram (SDD) is proposed to define the mutually exclusive relations between users or subjects and a set of roles. Finally, the Role Assignment Diagram (RAD) has been proposed to define the association between a user or subject and roles.

3.1. Role Diagrams

Role Diagrams (RD) are a special case of the well known UML Class Diagrams [12], where the classes are roles joined by generalization relations. The purpose of RD is to model the static structure of roles and to show the role graphical view for a given domain. The proposed language can describe roles and dependences, allowing the administrator to vary the degree of control over the order in which the user can use resources. By using the role hierarchy we optimize the permission definition, avoiding useless redundancies. We give common permissions to the role at the basis of the hierarchy. The language symbols are shown in Fig. 4. The first symbol (Fig. 4A) represents a generic role in visual policy access definition. The role object has a label representing the role name. The generalization relation object is represented by the arrow of Fig. 4B. The symbol is used between any two roles; the one at the tail of the arrow inherits the privileges of the one at head.

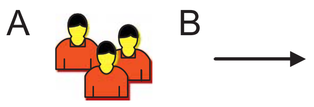


Fig. 4. The visual language tokens. (A) Role Object. (B) Generalization relation.

Fig. 5 shows a visual sentence describing an example of access policy roles in a University portal. The basic role is the Guest from which all others roles inherit privileges.

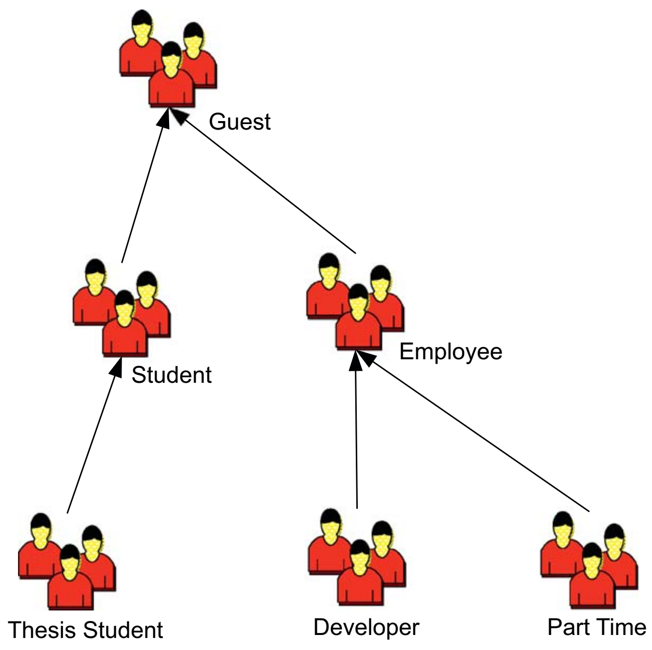


Fig. 5. Role Diagram sentence.

Fig. 6 shows the XPG grammar [11] of the Role Diagram.

3.2. Permission Diagrams

The Permission Diagram (PD) has been introduced to facilitate the definition of the access permission, which is a critical task in policy definition. It supports security administrators during the design and implementation of a policy, providing means for describing the use of a resource by a role. The PD language also allows security administrators to express restrictions on the use of the resource. Policies in the PD language context can be expressed as follows:

If ⟨*condition(s)*⟩ *then* ⟨*grant access to resource(s)*⟩

Generally, conditions are based on:

- time and/or date;
- user's or group's name;
- application;
- source and destination (for instance a source or destination network address);

The resources on which the administrator can define policies depend on the application domain. For instance, in the network service domain the resources may concern the use of:

- band;
- Virtual Private Network (VPN);

(1) $\mathrm{RD} \rightarrow \mathrm{ROLE\_OBJECT}$
$\Delta\,(\mathrm{RD}_1 = \mathrm{ROLE\_OBJECT}_1\,;\ \mathrm{RD}_2 = \mathrm{ROLE\_OBJECT}_2)$

(2) $\mathrm{RD} \rightarrow \mathrm{RD}'[2\_1]\mathrm{GENERALIZATION}[2\_1]\ \mathrm{ROLE\_OBJECT}$
$\Delta\,(\mathrm{RD}_1 = \mathrm{RD}'_1\,;\ \mathrm{RD}_2 = \mathrm{RD}'_2)$
$\Gamma\,\{(\mathrm{PLACEHOLD};\ |\mathrm{ROLE\_OBJECT}_2| > 0\,;\ \mathrm{PLACEHOLD}_1 = \mathrm{ROLE\_OBJECT}_2 - \mathrm{GENERALIZATION}_2\}$

(3) $\mathrm{RD} \rightarrow \mathrm{RD}'\textbf{<any>}\ \mathrm{PLACEHOLD}$
$\Delta\,(\mathrm{RD}'_1 + \mathrm{PLACEHOLD}_1;$
$\mathrm{RD}'_2 + \mathrm{PLACEHOLD}_2)$

Fig. 6. Role Diagram XPG grammar.

- File Transfer Protocol (FTP);
- Hyper Text Transfer Protocol (HTTP);

In general, in any application domain a resource is requested from a subject, such as a user, an application, or a service. The subject can make several actions on the released resource. Thus, we can describe a policy as the triple

$\langle subject, resource, action\rangle$

Basically, in order to define how a subject can exploit a resource some restriction on its use should be imposed. The PD language enables an administrator to define several uses of the network resources, specifying restrictions such as time, network bandwidth, and so on. Moreover, since the visual language is conceived to model role based policies, the subjects have to be intended as elements belonging to a role. Hence, the policies can be considered as:

$\langle Role\rangle$ use $\langle resource\rangle$ if $\langle restriction\rangle$ valid

The permission definition in the PD language follows that kind of representation. In order to define permissions, the administrator uses the symbols shown in Fig. 7. The PD language has three different symbols and a link. The first symbol (Fig. 7A) represents a role in the diagram describing the resource access. It has a tag representing the role name. The subjects can use all the resources that are joined with its role by arrows (Fig. 7C).

The arrow has a label representing the type of action that the subject can make on the resource. It is worth noting that in order to represent a resource the PD language uses the Generic Resource symbol (Fig. 7B), which can be specialized for each resource described in the language sentence. In this way the PD sentences become more readable and flexible. Finally, to represent restrictions on the use of a resource we use the Generic Restriction symbol (Fig. 7D). Role subjects cannot use a resource when an interdiction symbol is associated to the link connecting the role to the resource. Notice that multiple restrictions can be specified on the use of a given resource. Similarly to the Generic Resource symbol, also the restriction symbol is customizable in order to adapt the PD language to several permission access contexts.

An example of visual policy from the PD language is shown in Fig. 8. The policy depicts an example of resource

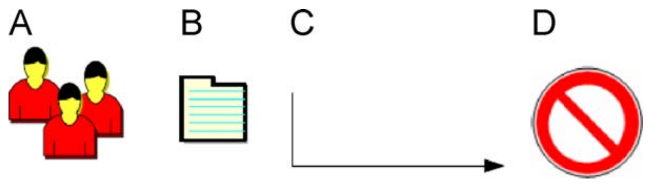


Fig. 7. The visual language tokens. (A) Role object. (B) Generic resource. (C) Action relation. (D) Generic restriction.

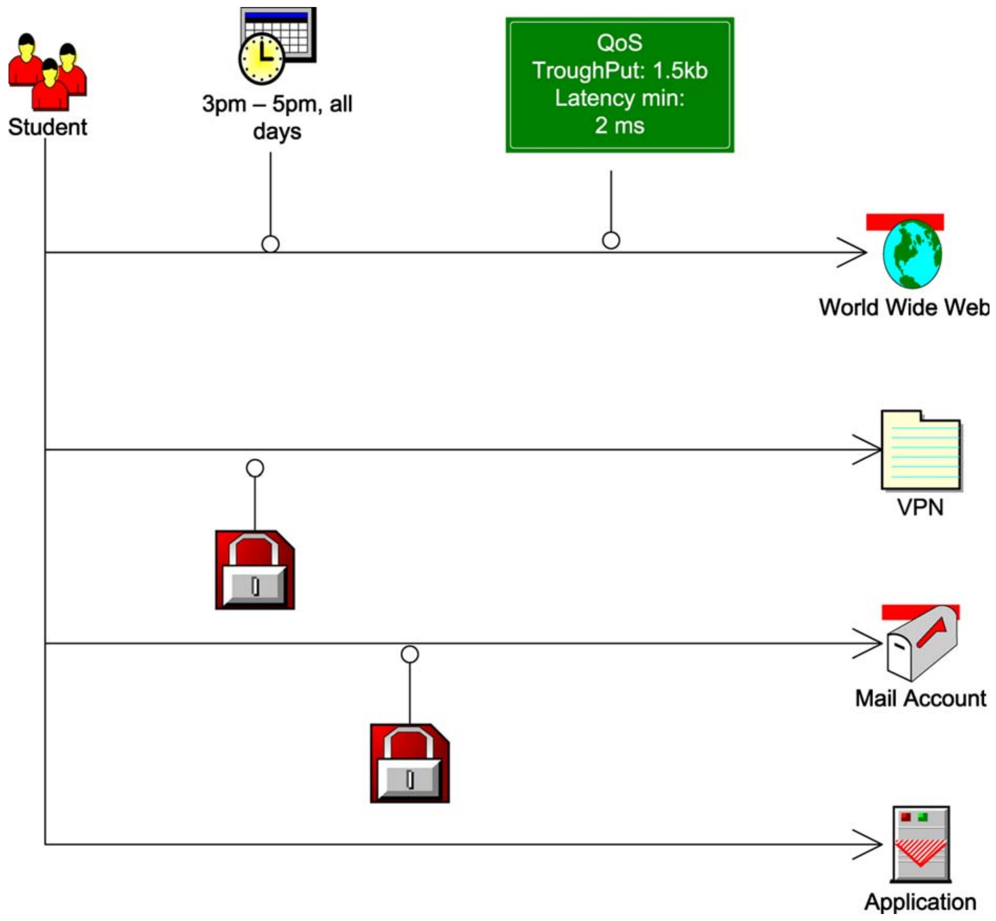


Fig. 8. PD language sentence.

access in the University context. The subject of the policy is the Student who can use the resources WWW, VPN, e-mail, and an Application provided by the University portal. Application can be used by the subjects of the Student role without restrictions. Typical examples of services enjoyable from the student in a University portal could be forum, chat, course timetable, and so on. Restrictions are defined on the other resources. The e-mail and the VPN are accessible from the subject in the Student role only with the appropriate certificate. Instead, the use of the Web is allowed only from 3 pm to 5 pm, and the bandwidth is limited to 1.5 Mbit.

In what follows, an example of XACML specification of permissions is provided. This also shows how the proposed visual languages simplify the RBAC policy specification, which would otherwise be a complex task if directly using XACML.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="urn:oasis:names:tc:xacml:1.0:context"
    PolicyId="SelectorPolicy"
    RuleCombiningAlgId="urn:oasis:names:tc:
    xacml:1.0:rule-combining-algorithm:permit-overrides">
    <Description></Description>
    <PolicyDefaults>
        <XPathVersion>
            http://www.w3.org/TR/1999/Rec-xpath-19991116
        </XPathVersion>
    </PolicyDefaults>
    <Target>
    <Subjects>
        <AnySubject/>
    </Subjects>
<Resources>
    <Resource>
        <ResourceMatch
        MatchId="urn:oasis:names:tc:xacml:1.0:
        function:regexp-string-match">
        <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">
        http://localhost:8080/opencms/opencms/system/galleries/video</AttributeValue>
        <ResourceAttributeDesignator
            DataType="http://www.w3.org/2001/XMLSchema#string"
            AttributeId="urn:oasis:names:tc:
            xacml:1.0:resource:resource-id"/>
        </ResourceMatch>
    </Resource>
</Resources>
<Actions>
    <Action>
        <ActionMatch
        MatchId="urn:oasis:names:tc:xacml:1.0:
        function:string-equal">
        <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">
```

```
            read
            </AttributeValue>
            <ActionAttributeDesignator
            DataType="http://www.w3.org/2001/XMLSchema#string"
            AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
            </ActionMatch>
        </Action>
</Actions>
</Target>
<Rule RuleId="AccessIfInGroup" Effect="Permit">
<Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
<SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"/>
</Apply>
<AttributeSelector RequestContextPath="//ResourceContent/Record/User/Cell/text()"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
</Apply>
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
<SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
AttributeId="login"/>
</Apply>
<AttributeSelector RequestContextPath="//ResourceContent/Record/User/Login/text()"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
</Apply>
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
<SubjectAttributeDesignator
DataType="http://www.w3.org/2001/XMLSchema#string" AttributeId="password"/>
</Apply>
<AttributeSelector RequestContextPath="//ResourceContent/Record/User/Password/text()"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
</Apply>
</Condition>
</Rule>
</Policy>
```

Finally, Fig. 9 shows the XPG grammar of the Permission Diagram.

3.3. *Separation of Duties Diagram*

The RBAC model allows us to define static restrictions of a subject to a role. For example, it can ensure that subjects cannot be members of both the purchasing role and the approving role. That is how static separation of duties ensures that the same person cannot purchase and approve the purchase within an e-commerce system. That restriction is specified in the proposed method by using the Separation of Duties Diagram (SDD). The administrator can exclude a user or a group of users from a role by appropriately arranging the symbols of the RD language (Fig. 10). The role (Fig. 10C) previously defined in the RD language can be connected to the Subject Symbol (Fig. 10A) through the arrow in Fig. 10B. The administrator selects the set of users in the Subject Symbol, which are consequently banned from the specified role. Fig. 11 shows an example of sentence from the RD language. The sentence specifies that the user in Group Beta cannot be in both Student and

$$\begin{array}{ll}
(1) & \text{RD\_SENTENCE} \rightarrow \text{ROLE\_OBJECT [1\_1] Action} \\
 & \Delta(\text{RD\_SENTENCE}_1 = \text{ROLE\_OBJECT}_1) \\
 & \\
(2) & \text{Action} \rightarrow \text{Action' [3\_1] RESOURCE } [2\_1]^{-1}\text{CONNECTOR [2\_1] RESTRICTION} \\
 & \Delta(\text{Action}_1 = \text{Action'}_1, \text{Action}_2 = \text{Action'}_2, \text{Action}_3 = \text{Action'}_3) \\
 & \Gamma(\text{PLACEHOLD}; |\text{RESTRICTION}_1 > 0|; \text{PLACEHOLD}_1 = \text{RESTRICTION}_1) \\
 & \\
(3) & \text{Action} \rightarrow \text{Action' [3\_1] RESOURCE }^{1} \\
 & \Delta(\text{Action}_1 = \text{Action'}_1, \text{Action}_2 = \text{Action'}_2, \text{Action}_3 = \text{Action'}_3) \\
 & \Gamma(\text{PLACEHOLD}; |\text{RESOURCE}_1 > 0|; \text{PLACEHOLD}_1 = \text{RESOURCE}_1) \\
 & \\
(4) & \textit{Action} \rightarrow \textit{Action'} <\textbf{any}> \textit{PLACEHOLD} \\
 & \Delta(\textit{Action}_1 = \textit{Action'}_1 + \textit{PLACEHOLD}_1; \\
 & \textit{Action}_2 = \textit{Action'}_2 + \textit{PLACEHOLD}_2; \\
 & \textit{Action}_3 = \textit{Action'}_3 + \textit{PLACEHOLD}_3;)
\end{array}$$

Fig. 9. Permission Diagram grammar.

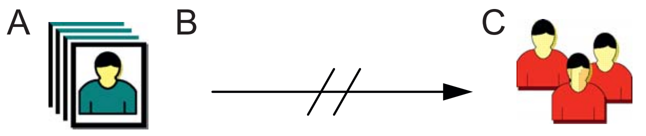


Fig. 10. Separation of Duties Diagram objects. (A) Subject symbol. (B) Exclusion symbol. (C) Role symbol.

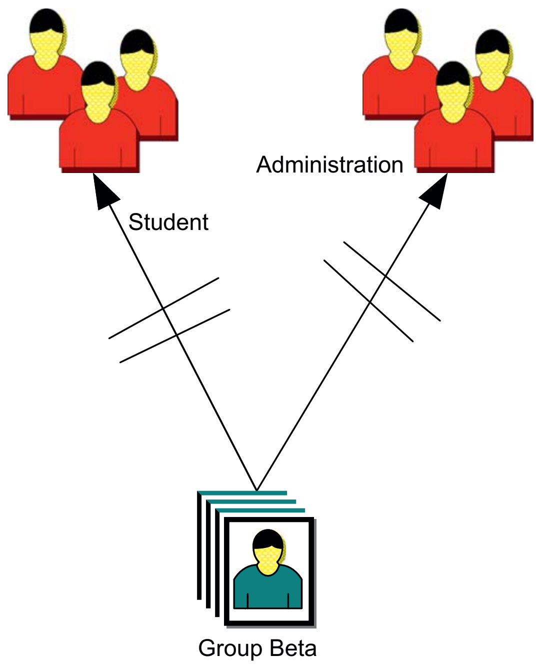


Fig. 11. Separation of Duties Diagram sentence.

Administrator roles. The subject organized in groups has to be selected from a centralized repository, previously populated by the administrator. The repository is also used by the Role Assignment Diagram, which is shown in the following subsection. It is worth noting that the definition of such restrictions on a subject is not mandatory.

$$\begin{array}{ll}
(1) & \text{SDD} \rightarrow \text{S} \\
 & \Delta(\text{SDD}_1 = \text{S}_1) \\
 & \\
(2) & \text{SDD} \rightarrow \text{SDD'[1\_2]E[2\_1]R} \\
 & \Delta(\text{SDD}_1 = \text{SDD'}_1) \\
 & \Gamma\{(\text{PLACEHOLD}; |\text{R}_2| > 0; \text{PLACEHOLD}_1 = \text{R}_2 - \text{E}_2\} \\
 & \\
(3) & \text{RD} \rightarrow \text{SDD'} <\textbf{any}> \text{PLACEHOLD} \\
 & \Delta(\text{SDD'}_1 + \text{PLACEHOLD}_1)
\end{array}$$

Fig. 12. Separation of duties grammar.

Fig. 12 shows the XPG grammar of the Separation of Duties Diagram.

3.4. *Role Assignment Diagram*

In the definition of policies it is worth defining for each session the subjects of a role that can use a resource. Thus, the administrator defines groups of subjects belonging to a role and expresses conditions for each of them on the use of a resource that had been previously put in relation with such role through the PD language. This type of policy is managed by a specific entity, which is meant as a module being able to process a Role Assignment Policy compliant with the used access control standard. The Role Assignment Policy contains information to determine which subjects are authorized to be in a role and what are the conditions to satisfy. The Role Assignment Diagram (RAD) has been defined to model that kind of policy.

The policies are modelled in the RAD language by triples:

$$\langle \textit{Subject}, \textit{Role}, \textit{Condition} \rangle$$

In particular, by using arrows (Fig. 13C) the administrator associates predefined roles (Fig. 13A) with subjects or group of subjects (Fig. 13B). The association says that the subject belongs to the role. In order to introduce membership conditions, one or more Restriction Symbol (Fig. 13D) can be used. This symbol is generic, and can be customized by the administrator to improve RAD expressiveness and readability. The introduction of new symbols representing policy conditions requires that the administrator define XACML functions implementing such conditions. It is worth noting that analogously to the RD language the subjects are selected from a centralized repository.

The sentence in Fig. 14 shows an example of sentence from the RAD language. Notice that the user Max belongs to the Administrator role without restrictions, and that the Group Alpha is enabled to be in the Employee role from the 9 am to 5 pm by using a customization of the Restriction Symbol.

Fig. 15 shows the XPG grammar of the Role Assignment Diagram.

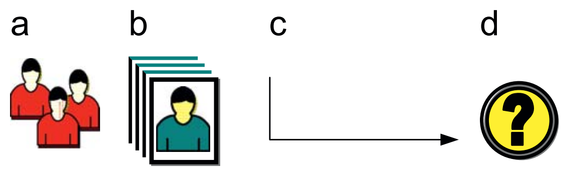


Fig. 13. RAD language objects: (A) Role symbol. (B) Group symbol. (C) Relation. (D) Generic restriction.

4. System prototype

In this section we present the Visual Policy Editor (VPE) embedding the visual languages presented above. It integrates several programming environments and software modules for composing, managing, generating, and deploying policies according to the proposed approach. Moreover, a version of the VPE has been implemented as an Eclipse plugin.

The layered architecture of the VPE is shown in Fig. 16. Its main modules are: Visual Environment, Symbol Editor, Diagram Representation, Policy Generator, and Policy Deployment Module.

4.1. Visual environment

The Visual Environment is the main module of the VPE (see Fig. 17). For each defined visual language it provides a palette containing symbols and links. By using the drag

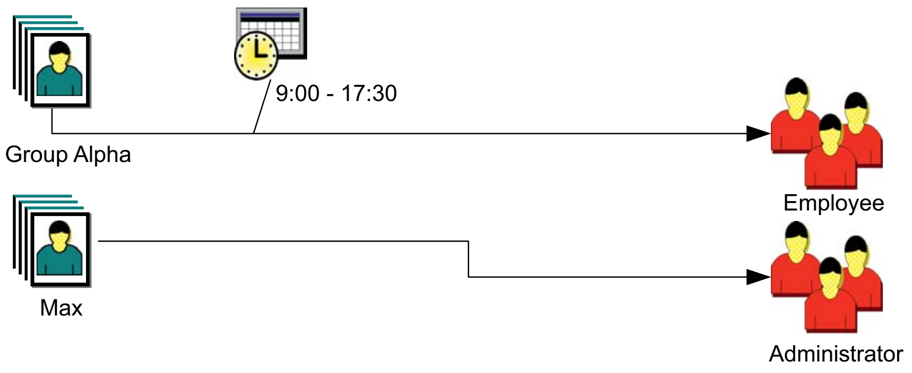


Fig. 14. RAD sentence.

(1) RAD→ GROUP_SYMBOL [1_1] RelationSymbol
Δ(RAD$_1$ = GROUP_SYMBOL$_1$)

(2) RelationSymbol → RELATION_SYMBOL' [3_1] ROLE_SYMBOL [2_1]$^{-1}$CONNECTOR[2_1] RESTRICTION_SYMBOL
Δ(RelationSymbol$_1$=RelationSymbol'$_1$;RelationSymbol$_2$=RelationSymbol'$_2$; RelationSymbol$_3$ = RelationSymbol'$_3$)
Γ(PLACEHOLD; | RESTRICTION_SYMBOL$_1$ >0 | ; PLACEHOLD$_1$ = RESTRICTION_SYMBOL$_1$)

(3) RelationSymbol → RELATION_SYMBOL' [3_1] ROLE_SYMBOL^1
Δ(RelationSymbol$_1$=RelationSymbol'$_1$;RelationSymbol$_2$=RelationSymbol'$_2$; RelationSymbol$_3$ = RelationSymbol'$_3$)

(4) RelationSymbol →RelationSymbol ' <**any**> *PLACEHOLD*

Δ(RelationSymbol$_1$ = RelationSymbol '$_1$ + *PLACEHOLD*$_1$;
RelationSymbol$_2$ = RelationSymbol '$_2$ + *PLACEHOLD*$_2$;
RelationSymbol$_3$ = RelationSymbol '$_3$ + *PLACEHOLD*$_2$)

Fig. 15. Separation of duties grammar.

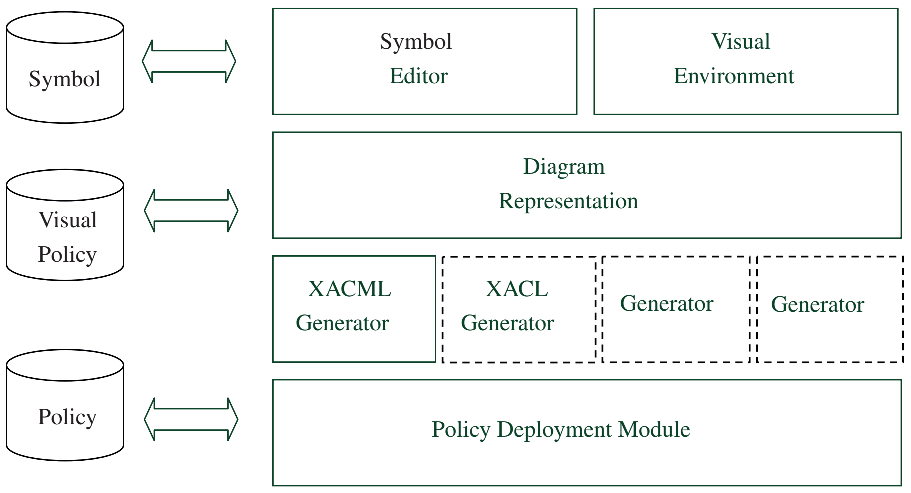


Fig. 16. Architecture of the visual policy editor.

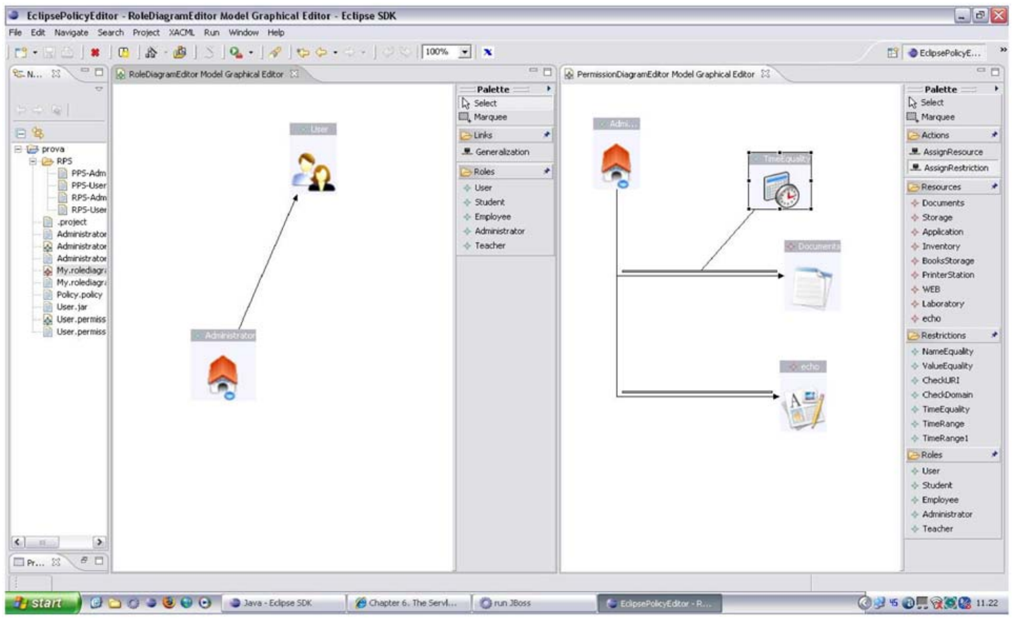


Fig. 17. The eclipse based visual policy editor.

and drop metaphor the administrator places the language symbols on the workplane and joints them through connectors. Visual sentence composition is syntactically driven by the editor. Before generating the XACML policy, the visual language compiler embedded in the VPE executes a complete syntactic control on the sentence, presenting errors to the user through a log in a dialog window, also providing support to solve them.

4.2. *Symbol editor*

As shown above, users can extend or customize the proposed visual languages by drawing or specializing visual symbols. Thus, this module is crucial for the personalization of the approach and the customization of the tool for different application domains. This component provides features to draw a new symbol or to customize an existing one. The personalization of the graphical layout of a symbol allows us to make its meaning more intuitive for the given application context. The Symbol Editor supports the creation and the personalization of language symbols, and also the import of those created through other visual editors. The symbol personalization is determined by assigning them a suitable semantics. The administrator can define symbol semantic specifying resources and constraints or eventually choosing symbol semantics from a set of predefined ones.

4.3. *Diagram representation*

In order to store visually specified policies the system uses two kinds of representations: graphical based on SVG (Scalable Vector Graphics) [13], and semantics based on

XML. These two kinds of representations are first used to check the sentence syntax, and then to generate the XACML policy through the Policy Generator. This abstraction may allow us to implement several modules in order to generate policies based on RBAC, as shown in the system architecture of Fig. 16. Using that intermediate representation as input, several RBAC based policy implementation tools can be provided by us or other vendors. It is worth noting that the use of SVG allows us to compose visual sentences through any graphical editor, and successively import them in the Visual Policy Editor prototype to generate XACML policies.

4.4. *Policy generator*

The Visual Production Editor supports several RBAC based Policy Generator engines. In this work we propose one for translating the intermediate representation of visual RBAC policies into XACML policies. The XACML Generator generates XACML policies starting from the Diagram Representation. Basically, the aim of this module is to fill XACML templates by using the information and semantics introduced through the system user interface.

Starting from the policy visual representations, the Policy Generator produces the following set of XACML policies [7]:

- *Role PolicySet* associates a user of a role with a set of permissions.
- *Permission PolicySet* contains the permissions associated to a given role, the resource description, and the actions allowed for the associated user. The Permission PolicySet can also contain other kinds of access restrictions such as, date, time, and possibly other associations for permissions inherited by the role hierarchy.
- *Separation of Duties PolicySet* defines restrictions on the role set that can be used by a given user.
- *Role Assignment PolicySet* defines the roles that can be associated or enabled on a given user.

4.5. *Policy deployment module*

The Policy Deployment Module is used to deliver XACML policies generated by the Visual Policy Editor. It activates a secure communication between the proposed system and the Policy Repository by using authentication and communication means. The security of the policy delivery schema is based on secure authentication and secure transmission scheme.

5. Case study

The system has been used experimentally in several application domains. In particular, it has been used in the context of collaborative environments, Voice Over Ip infrastructures, and multimedia content management platforms providing video streaming services, which is the one presented in this section.

In particular, we have cooperated with a software development company to develop a Multimedia Content Management System (MCMS) for news and forecasts, and have used our approach to embed access control policies within it.

The MCMS supports a wide range of users and devices, from PC clients to mobile phones, in a transparent way. It is capable of delivering content based on many factors, like multimedia device capabilities, bandwidth, and user credentials, without adding complexity to system administration and to the content creation process. Moreover, transparency is achieved by means of a fine grained access control mechanism.

The MCMS has been built upon OpenCMS, a well known Open Source Content Management System that can be easily extended by adding java modules. It stores text based contents by using XML so that it can be easily translated into different formats to support heterogeneous devices. OpenCMS supports WAP technology, hence mobile users can access the same text based content that is accessible to PC users in a transparent way. The same level of flexibility can be achieved for multimedia content, like image motion, audio, and video documents, thanks to streaming technology. Current streaming servers support multiformat streams, so that a given content can be streamed with different resolution and bandwidth requirements. Each content is associated with a different audit, each of which is a view of the stream. The selection of the right stream based upon device capabilities and user rights requires a fine grained access control mechanism. Moreover, also notification must be extended in order to better support mobile devices.

The goals just described have been reached by extending OpenCMS with new modules for video streaming content delivery, and SMS notification. Then, we have added support for role based policies by embedding a Policy Management System and a Visual Policy Editor based on our approach. Thanks to visual languages, the administration of users and authorizations has been simplified, making it accessible to unexperienced CMS administrators, which will most probably dislike computer complexity if they use a CMS to facilitate the management of content in complex web sites. The diagram in Fig. 18 shows the components of the case study system architecture and their relations.

The OpenCMS block is the Content Management System. As previously explained, the standard OpenCMS system was extended with three new modules:

- The News Module providing support for live news and forecast content.
- The Video Gallery providing support for Video Content stored on the streaming server Darwin [14].
- The Role Based Policy Module providing support for Policy Authorization control, which acts as a PEP.

In addition to the standard web-enabled and streaming capable clients we have added support for mobile phone

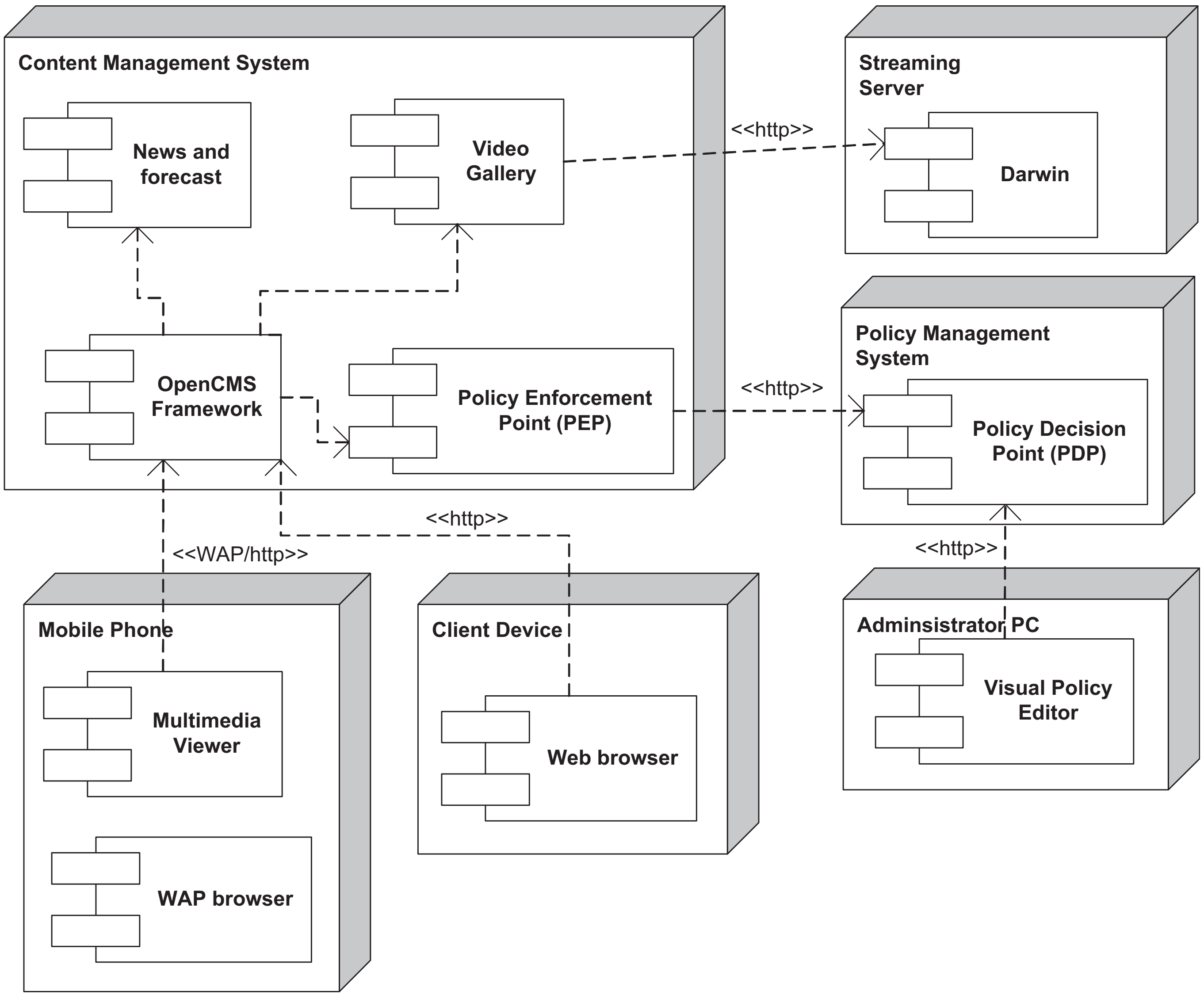


Fig. 18. Case study system architecture.

clients by writing a client application based on the Java 2 Micro Edition (J2ME) platform. The Policy Management System performs several functions, including policy configuration and authorization control. In this context the PEP is implemented like an extension module in OpenCMS. Moreover, the Visual Policy Editor has been customized with domain specific visual symbols. We have embedded several access policies within the MCMS in order to control access to resources. These are represented by forecasts that are supposed to be accessible in streaming from devices equipped with web browsers, and mobile phones supporting WAP technology. The following statements describe three examples of policies implemented within our multimedia CMS:

- Unregistered users can read textual news, and receive SMS notification no more than two times in a day.
- Registered users with a basic account have the same rights as unregistered users, but in addition they can stream media forecasts once a day.
- Registered users with a premium account have the same rights as basic user, but without limitations.

In order to implement them with our Visual Policy Editor we have started from their specification within the Role Diagram. In the scenario depicted in Fig. 19 we have defined five roles: Unregistered user, Registered user, Basic user, Premium user, and Content Manager. The Unregistered user is the basic role, and its permissions are common to all other roles, hence to any other user profile. Content Manager and Basic user have a common ancestor in the hierarchy, which means they share the same rights inherited from Unregistered user. After the Role Diagram we have specified the Permission Diagram, in which we have defined the permission assignment to roles. In the diagram depicted in Fig. 20 we define permissions for the role Unregistered user. In particular, the diagram represents the following statement: *Unregistered user can read textual news, and receive sms notification no more than two times a day*; the subsequent diagrams (Figs. 21

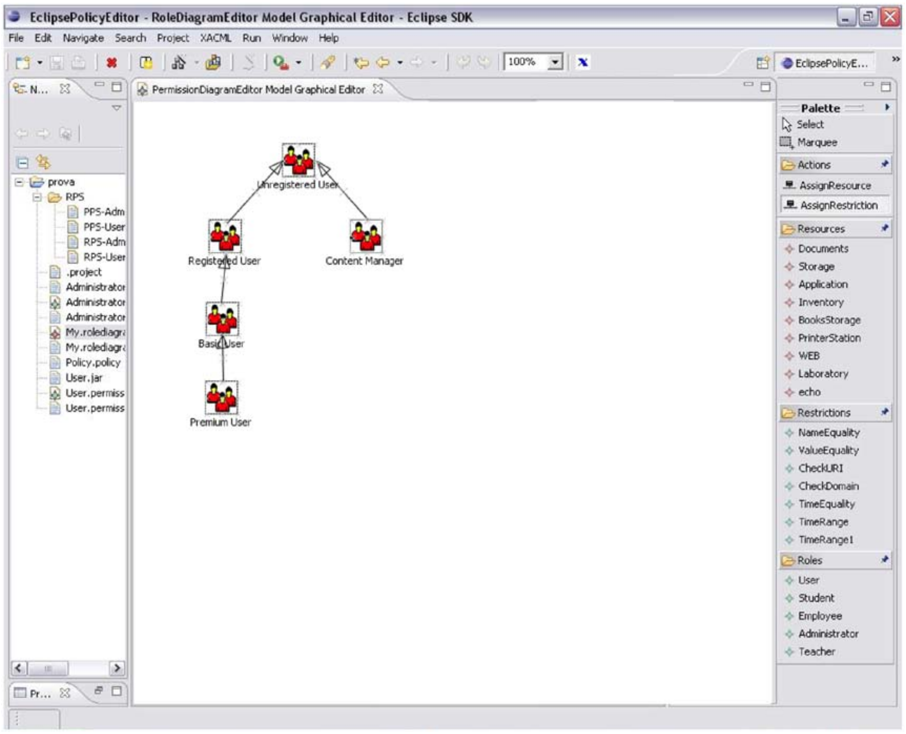


Fig. 19. The Role Diagram for the proposed policies.

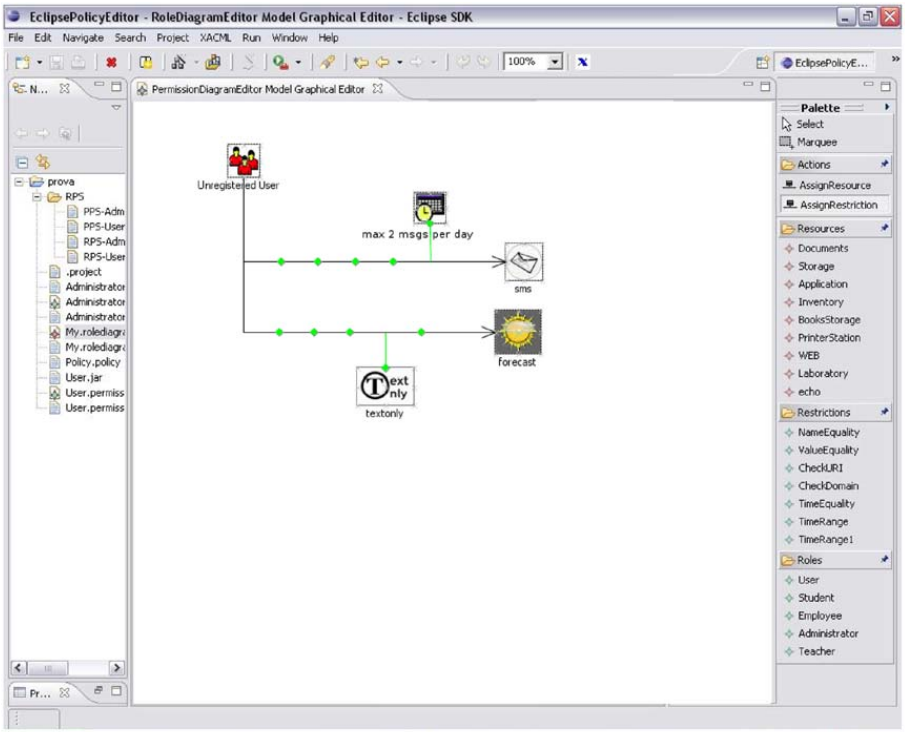


Fig. 20. Unregistered user Permission Diagram.

and 22) depict the following statements: *Registered user with a basic account has the same rights as an unregistered user, and in addition it can stream media forecast once a day* and *Registered user with a premium account has the same rights as Basic user, but they do not share the same limitations*; All the diagrams will be translated in XACML format, compressed in a JAR file, and deployed on the Policy Management System (PMS) via HTTP protocol.

Fig. 23 shows the web interface of the PMS. After the policies upload on the PMS, the client applications are enabled to invoke access authorization requests to the PDP embedded in the PMS. Interaction with PMS components is made via webservices technology. In

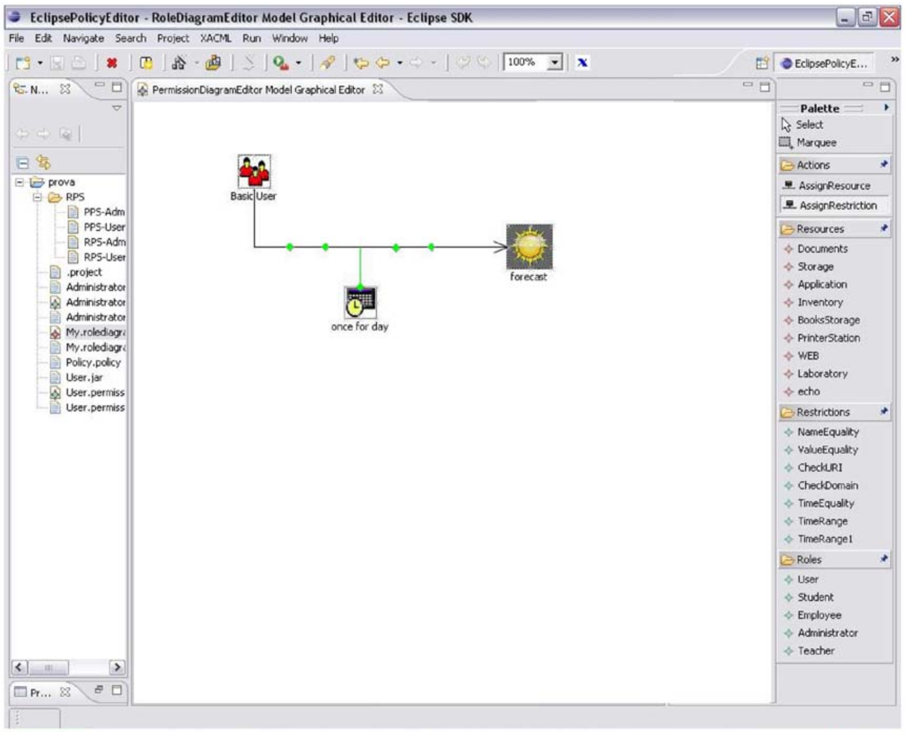


Fig. 21. Basic user Permission Diagram.

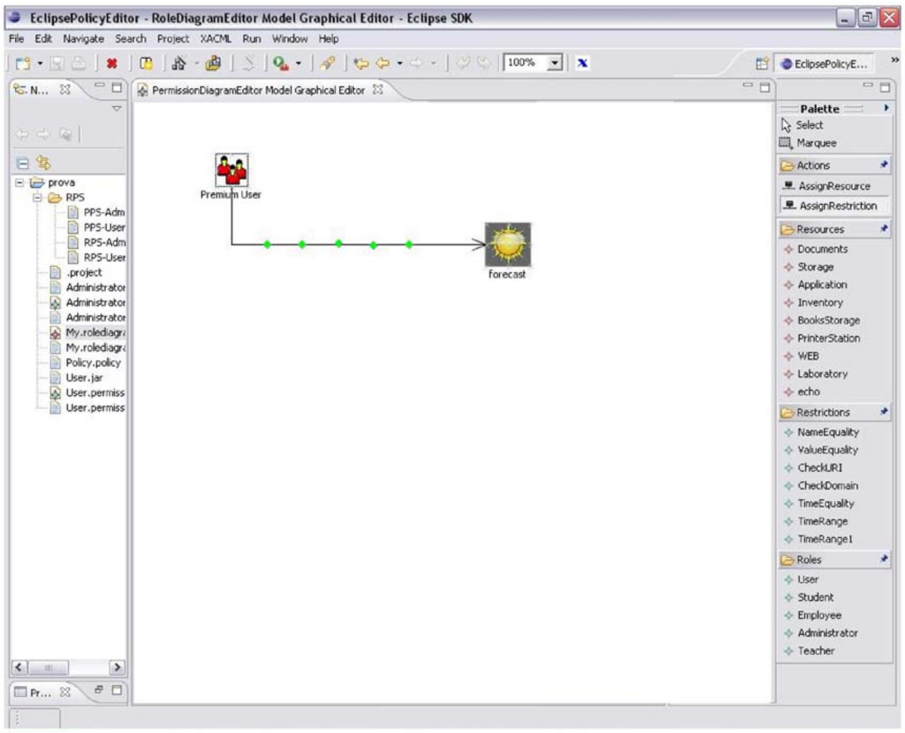
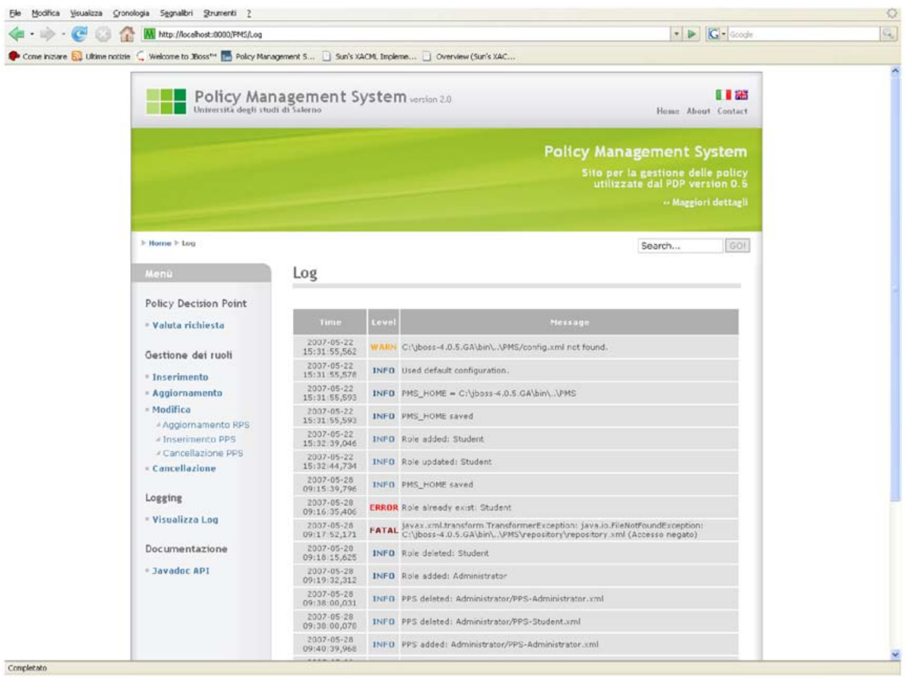

EclipsePolicyEditor - RoleDiagramEditor Model Graphical Editor - Eclipse SDK
File Edit Navigate Search Project XACML Run Window Help
PermissionDiagramEditor Model Graphical Editor
Premium User
forecast
Palette
Select
Marquee
Actions
AssignResource
AssignRestriction
Resources
Roles
Restrictions

Fig. 22. Premium user Permission Diagram.

order to benefit from this service infrastructure each application must interact with the PEP component to enforce the policy and request authorization to the PDP. As said above, we have integrated a PEP module within OpenCMS so that all the requests made to the OpenCMS application will be translated in XACML format and transmitted to the PDP embedded in the PMS. Moreover, in order to enable streaming capabilities in mobile devices we have also developed a client application with the Java Micro Edition, which can be deployed on a wide range of mobile phones.

All the software components have been realized with Java Technologies. PMS uses Tomcat and Axis for web interface and web services support. Policy creation and



Log

Time	Level	Message
2007-05-22 15:31:55,562	WARN	C:\jboss-4.0.5.GA\bin\..\PMS/config.xml not found.
2007-05-22 15:31:55,578	INFO	Used default configuration.
2007-05-22 15:31:55,593	INFO	PMS_HOME = C:\jboss-4.0.5.GA\bin\..\PMS
2007-05-22 15:31:55,593	INFO	PMS_HOME saved
2007-05-22 15:32:39,046	INFO	Role added: Student
2007-05-22 15:32:44,734	INFO	Role updated: Student
2007-05-28 09:15:39,796	INFO	PMS_HOME saved
2007-05-28 09:16:35,406	ERROR	Role already exist: Student
2007-05-28 09:17:52,171	FATAL	javax.xml.transform.TransformerException: java.io.FileNotFoundException: C:\jboss-4.0.5.GA\bin\..\PMS\repository\repository.xml (Accesso negato)
2007-05-28 09:18:15,625	INFO	Role deleted: Student
2007-05-28 09:19:32,312	INFO	Role added: Administrator
2007-05-28 09:38:00,031	INFO	PPS deleted: Administrator/PPS-Administrator.xml
2007-05-28 09:38:00,078	INFO	PPS deleted: Administrator/PPS-Student.xml
2007-05-28 09:40:39,968	INFO	PPS added: Administrator/PPS-Administrator.xml

Fig. 23. The web interface of the PMS.

basic evaluations are made through an open source API for XACML. The Visual Policy Editor is mainly written with Swing API and JDOM, and it exploits XACML for policy generations. Finally, the mobile phone client is written with JME.

6. System usability

We selected two usability experts to conduct a pilot test before the proposed usability test. The experts, who were not in the development team of the tool, worked independently using Nielsen's heuristic evaluation [15], and then discussed until they reached an agreement. Once the main usability problems were identified and fixed, the proposed usability test was carried out through a one-to-one session (i.e., with a supervisor for each subject) using the think aloud technique. A group of heterogeneous subjects was recruited, including managers and technicians from industries and universities, whose profiles are summarized in Table 1. In particular, first and second columns contain the ID of the subject and his/her background, while the third column reports his/her gender.

6.1. Experiment design

All the subjects underwent an introductory course of 60 min on the system and its visual notation. Successively, these subjects have been asked to use the tool for 20 min with the possibility of invoking tutor support. After that, they were asked to use the system to define access policies on a usage scenario they were familiar with. For example, one of the subjects (i.e., a Manager of a partner company), who had experience on the Voice Over IP, decided to define and model access polices on his company communication infrastructure.

Table 1
Subject' background.

Subject ID	Background	Sex
ID1	Computer Science Lecturer	F
ID2	Master Student in Mathematics	F
ID3	Master Student in Mathematics	M
ID4	Master Student in Computer Science	M
ID5	Master Student in Computer Science	F
ID6	Manager of a partner company	M
ID7	Manager of a partner company	M
ID8	Practitioner of a partner company	F
ID9	Practitioner of a partner company	M
ID10	Enterprise Manager	M
ID11	Research Fellow in Computer Science	M
ID12	Research Fellow in Mathematics	M
ID13	Computer Science Lecturer	M
ID14	Assistant Professor in Mathematics	F
ID15	Assistant Professor in Computer Science	M
ID16	PhD Student in Mathematics	F
ID17	PhD Student in Mathematics	M
ID18	PhD Student in Mathematics	F
ID19	PhD Student in Mathematics	F
ID20	PhD Student in Mathematics	F

The recruited subjects were volunteers with different backgrounds (see Table 1) and professional experiences. However, the majority of them did not master the XACML language. This was especially true for the subjects without computer skills, who could not define and model access polices without using the proposed visual language based system.

The subjects carried out the experiment without having the possibility of invoking individual tutor support. After accomplishing the task, they were asked to fill in a questionnaire to provide information on the usability

Table 2
Questions of the usability questionnaire.

Category	Id	Question
General evaluation	Q.2.1	The tool provides a nice user interface
	Q.2.2	Using the tool is simple
	Q.2.3	The aroused feeling by the tool use is satisfactory
Special judgment	Q.3.1	The user interface is pleasant
	Q.3.2	The tool is simple to use
	Q.3.3	The tool proposes specific error messages
Tool learning	Q.4.1	Learning to use the tool is simple
	Q.4.2	The required time to use the tool is appropriate
	Q.4.3	Remembering the commands and their use is simple
	Q.4.4	The number of steps to carry out a task is appropriate
	Q.4.5	The required time to insert a new role or resource is appropriate
	Q.4.6	The number of steps to insert a new role or resource is appropriate
	Q.4.7	Exporting the policies created using the tool is simple
Information grant	Q.5.1	Icon names and objects have a clear meaning
	Q.5.2	Each set of operations produces a predictable result

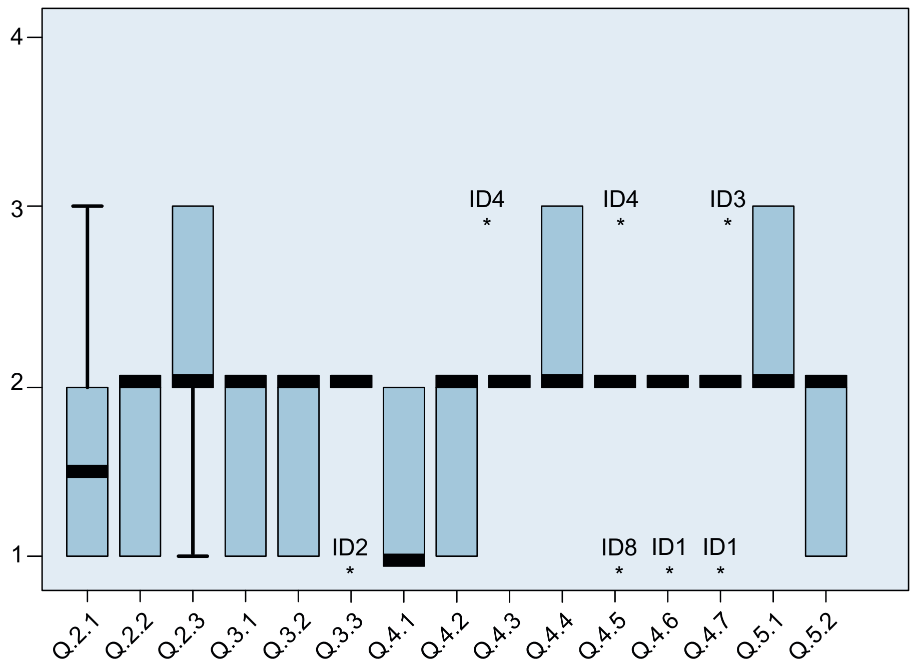


Fig. 24. The boxplots for the answers of the usability questionnaire.

they perceived. The questions composing the usability questionnaire were organized into five categories (see Table 2). Subjects expertise and their general reaction in terms of satisfaction degree were evaluated through questions in the categories Subject Background (not shown in the table) and General Evaluation, respectively. Questions in the Special Judgment category aimed at assessing the perceived usability with respect to the graphical user interface. The Tool Learning category aimed at evaluating the satisfaction degree to master the tools. Finally, the information provided by the tool during its usage was evaluated through the questions in the Information Grant category. All these questions of expected closed answers according to a Likert scale [16]: 1 (strongly agree), 2 (agree), 3 (neutral), 4 (disagree), and 5 (strongly disagree).

6.2. Results

The data collected from the usability questionnaire are visually summarized in the boxplots of Fig. 24. This figure shows a good distribution of the answers for each question of the questionnaire.

The General Evaluation was fairly good for the majority of the subjects, as the boxes for questions Q.2.1, Q.2.2, and Q2.3 reveal. Nearly the totality of the subjects had a good reaction concerning the system usage, as the boxes for questions Q.3.1, Q.3.2, and Q3.3 show. However, the subject ID2 expressed a better judgment on the error messages proposed by the tool (see Q.3.3 box). Moreover, on the perceived simplicity of the tool learning a good agreement has been achieved. Indeed, on the questions Q.4.4 the subjects expressed a worse judgment.

However, as the box for question Q.4.4 shows, the number of steps required to accomplish a task was considered appropriate.

A better agreement on these questions could be probably achieved by using a different graphical toolkit. In fact, the tool was implemented as an Eclipse plug-in, but the Eclipse graphical toolkit imposes some constraints in the development of the plug-in graphical user interface and in the operations to perform in the accomplishment of a particular task. Finally, also on the Information Grant category a good agreement level has been achieved.

6.3. Discussion

The usability study revealed that the satisfaction degree of the subjects is more than sufficient. Subjects judged the hierarchy of visual languages sufficiently intuitive. Moreover, they also appreciated the possibility to define new roles, and the simplicity in associating new software and hardware resources to them. Particularly appealing was judged the possibility of rapidly turning visual policies into XACML policies. A subject found troubleshooting incorrect policies difficult. On the other hand, all subjects grew confident with the system after the introductory course. Concerning the effort required to understand the Visual Policy Editor, the majority of the subjects understood it quickly. No problems were manifested in the comprehension of the hierarchy of the proposed visual languages. Nevertheless, two subjects with low expertise in Computer Science (i.e., the ones with mathematics background) required a moderate effort to understand the hierarchical visual language organization.

The Role Diagram required more effort than the second tier visual languages. In particular, we observed that three subjects raised some issues to understand the RD visual language. Probably, this was also due to the fact that subjects were not familiar with software engineering or object oriented programming languages, hence mapping the concept of inheritance between roles was not immediate to them. Nevertheless, encouraging results were achieved in terms of time spent to compose visual sentences. On the average, users spent 15 min completing a visual sentence in both the upper and second tier visual languages. We did not account for the minutes required to define the symbols of the visual languages.

In most cases visual languages improve productivity of expert and non-expert users as they are easier to learn than textual languages [17]. Nevertheless, this is not always true, hence the usefulness of visual languages in a specific domain needs to be assessed and verified [18]. In our case the data analysis has revealed that the recruited subjects perceived a good usability degree of the visual language based tool. To generalize this result, a further investigation using a larger dataset is required. However, the primary goal here is to propose a visual tool and companion methods to model policy according to RBAC.

7. A comparative evaluation

In this section we present a comparative evaluation between our tool and one of the most pertinent competitor, namely the XGrids Tools [19], which also uses a visual based approach.

7.1. Context

The comparative study was carried out by using a one-to-one session using the think aloud technique. Similarly to the usability study discussed above, we recruited managers and technicians from industries and universities, whose profiles are summarized in Table 3. The experiment was organized in three days in order to compare the efficacy of our visual formalism with respect to the XGrid approach. Due to the unavailability of tools implementing the XGrid approach, we have implemented a prototype in order to conduct the comparative study presented here.

7.2. Experiment design

All the subjects underwent an introductory course of 90 min on the investigated systems and theirs visual notations. The subjects were then asked to use the tools for 20 min with the possibility of invoking tutor support. After that, they were asked to use the systems in the definition of access policies for three different usage scenarios or tasks:

- T1: File Hosting Service: In the first test scenario we asked users to model access policies for a typical Internet hosting service specifically designed to offer "network storage" for personal backup, file access, or file distribution. Usually, in such a site, users can upload their files and share them publicly or keep them password-protected. Most online file storage services offer space on a per-gigabyte basis, and sometimes include a bandwidth cost component as well. Usually, these will be charged monthly or yearly. Under some conditions the service might be offered for free, relying on advertising revenue.
- T2: Massively multiplayer online role-playing game (MMORPG). In the second scenario we asked users to model policies for a computer role-playing game in which a very large number of players interact within a virtual game world. In role-playing games players

Table 3
Subject' background.

Subject ID	Background	Age
ID1	Computer Science Lecturer	24
ID2	PhD Student in Computer Science	25
ID3	Practitioner of a partner company	24
ID4	Manager of a partner company	40
ID5	Master Student in Computer Science	41
ID6	Administrative Accounter	50

assume the role of a fictional character (often in a fantasy world), and take control over many of that character's actions. During the game, the player acquires experience points, which are useful to gain new skills for his character. We asked users to model the system of rules specifying how and when a character might acquire new skills.
- T3: Research Labs: In the third scenario we asked users to model access control policies for a content management system in the context of a research laboratory, in which researcher, students, and employees are enabled to different levels of access to documents and portions of documents, stored in the CMS. In particular, any person is affiliated to a research group and any research group obey to different constraints in order to access a given document, or any of its sections.

The tasks had increasing difficulty. The subjects performed each task using both the tools according to their preferences. Once the tasks were accomplished the subjects were asked to fill in a post-experiment survey questionnaire (see Table 4) composed of eight questions in order to assess the overall quality of the provided material, the perceived usefulness of the experimented tools, and the clearness of the tasks. As shown in Table 4, the survey requires both closed and open questions.

At the end of the experiments the supervisor collected the post-experiment survey questionnaires and the defined policies, which were successively analyzed to get the answers provided by the subjects and to assess the overall quality of the policies. The supervisor also collected information regarding the performed tasks. Indeed, the time and the number of mistakes to accomplish the tasks.

Each subject was provided with the following material:

- Handouts of the introductory presentation.
- Printout of the tasks.
- Post experiment survey questionnaire.
- Some white sheets and a pencil.

The recruited subjects were volunteers with different backgrounds (see Table 3) and professional experiences. However, the majority of them did not master the XACML language.

7.3. Results

The experiment aimed at recording the time needed to carry on the XGrid model, and on the VPE Model (see Table 5 for details).

Users were divided in two groups (Group A and Group B). In group A any user has a good computer science skill, in group B, instead, we have users with a base level of computer science skill. For each group we have the following mean results (Table 6).

From the analysis of the time users spent on the test, and the answers they provided in the questionnaire, we found that the two systems have some common issues, but Visual Policy Editor is considered easier to understand, more complete, and easier to use.

Table 5
Minutes needed by each subject to complete the experiments.

Id	Minutes needed with XGrid	Minutes needed with VPE
S1	29	20
S2	36	29
S3	29	22
S4	20	34
S5	36	21
S6	25	33

Table 6
Average time in minutes.

Group	XGrid	VPE
A	≈ 31	≈ 23
B	≈ 27	≈ 29

Table 4
Questions of the comparative evaluation.

Category	Id	Question
General evaluation	Q.1.1	Is finding subject, resources and actions simple
Special judgment	Q.2.1	The user interface is pleasant
	Q.2.2	The tool is simple to use
	Q.2.3	The tool proposes specific error messages
Tool learning	Q.3.1	Learning to use the tool is simple
	Q.3.2	Do you have any problems using the Expandable Grid
	Q.3.3	Do you find the Expandable Grid appropriate for similar problems
	Q.3.4	Do you have any problems using the Visual Policy Editor
	Q.3.5	Do you have any problems defining role and role hierarchy using the Visual Policy Editor
	Q.3.6	Do you have any problems defining permissions using the Visual Policy Editor
	Q.3.7	Do you find the Visual Policy Editor appropriate for similar problems
	Q.3.8	Which tools did you find easier
	Q.3.9	Which tool is more suitable for an applicative domain similar to the ones proposed

Table 7
Answers for the open questions of the questionnaire.

Question	1 (strongly agree)	2 (agree)	3 (neutral)	4 (disagree)	5 (strongly disagree)
Q.1.1	4	1	1	0	0
Q.2.1	0	1	1	0	4
Q.2.2	4	0	0	0	2
Q.2.3	3	0	0	0	0
Q.3.1	2	2	1	0	2
Q.3.2	1	1	2	0	2
Q.3.3	1	1	3	0	1
Q.3.4	0	1	1	2	2
Q.3.5	0	0	0	2	4
Q.3.6	0	0	1	2	3
Q.3.7	0	0	1	2	3

In particular, the XGrid tool is considered by the users a conceptually easy to learn tool. It provides a uniform representation of access control policies, but the graphical representation of the grid does not fit well any application domain. The XGrid tool supports Hierarchical Relationships and Permission Inheritance, but it seems difficult to use when constraints needs to be modelled. Furthermore, it lacks flexibilities when policies needs to be rewritten or modified.

The VPE, as the XGrid tool, has been considered easy to learn and understand, and it supports Hierarchical Relationships and Permission Inheritance too. In addition, it looks more flexible and easier to use when policies need to be modified or rewritten, and it offers a better support to constraint definitions. Perhaps, it looks easier for computer aware users, probably due to the similarity between the visual languages adopted by the tool and many other visual languages used in other fields of computer science (see Table 7 for details).

In conclusion, the XGrid system provides a visualization metaphor enabling the user to have a complete view of the access control policies. It is simple and immediate, but sometimes the grid modelling is a complex task for some application domains, especially when several kinds of restrictions are required.

Visual Policy Editor, instead, suits better to various application domains and, as written so far, it resulted more flexible. On the other hand, its metaphor relies on four visual languages, and it does not support a whole look to policies.

7.4. *Threats to validity*

The threats to validity that could affect the study (i.e. internal, construct, external, and conclusions validity threats) are described in this section. Generally, the internal validity is only relevant in studies that try to establish a causal relationship. Thus, the internal validity threats are relevant for our study as we aimed at comparing our approach with *XGrids*. The internal validity is mitigated by the experiment design, since each subject worked on three different tasks and with two approaches. Furthermore, the results of the survey questionnaire revealed that the subjects found clear everything regarding the experimentation. Finally, the subjects knew neither the goals of the experiment nor its hypothesis.

The construct validity threats (i.e. the interactions between different treatments) were mitigated by a proper design of the experiment. In fact, depending on the experimented approach, the measurement of the dependent variable was performed by considering the times gathered by the supervisor. The results could also be affected by the method used to identify defects within the policy modelled by the subjects. Regarding the survey questionnaire, it was designed by using standard ways and scales.

External validity refers to the approximate truth of conclusions involving generalizations. This kind of threat is always present when students (Ph.D. students in Computer Science in our case) are used as subjects. However, they had a very good analysis, development, and programming experience. Due to their expertise, academic subjects are not far from professional programmers. We also used professionals as subjects to better investigate the benefits of using our approach to define access policies. To further confirm or contradict the achieved results, it will be worth replicating the experiment within different professional development environments. Let us also note that none of the recruited subjects abandoned the experiments.

It is also worth mentioning that not all threats to validity have been properly addressed, due to the nature of the investigation. For example, conclusion validity threats (i.e., the possibility to derive legitimate conclusions from observations) cannot be applied to our study since statistical tests are not used to reject experimental null hypotheses.

8. Related works

In the last two decades researchers have proposed several approaches for the definition and management of access and security policies, based on different metaphors [8,20–27]. Some of them adopt visual based metaphors to define and manage access and security policies. In particular, visual environments have been defined based on access matrices [20,21], annotated constraint graphs [28], task based authorization controls [22], multilayered security management [23], and roles [24,27].

Heydon et al. proposed the Mirò system [20], relying on the access matrix model to specify authorization policies. Two visual languages underlying this system are proposed: an instance language through which users could specify the access matrix and a constraint language through which they could specify possible restrictions of the underlying system on which access permissions should be enforced. These languages are based on hierarchical graphs, hence they are too technical for non-expert users. Moreover, it is difficult to adapt Mirò for modelling authorizations at high level, as required in most modern application domains, since it is mainly designed for file-system security. Based on the same underlying model, Hutchison and Saul presented a high-level security policy specification environment [21]. By means of a graphical interface the user receives a Message Sequence Chart (MSC) to design security protocols, which are evaluated in terms of performances using meta-execution facilities. The drawback of this system is its strong coupling with the underlying security model. On the other hand, MSCs appear friendlier than graph-based visual languages.

The Language for Security Constraints on Objects (LaSCO) uses an annotated constraint graph to visually specify authorization policies [28]. This language is event-driven and uses first order logic expressions to specify object constraints. An interesting characteristic of this language is the possibility of addressing system dynamics, giving the possibility to specify access policies in terms of system execution. This makes it suitable for a broader class of application domains. However, LaSCO lacks a metaphor-based paradigm to facilitate its use to non-technical people. Another policy language based on logic is the Authorization Specification Language (ASL) [25]. It introduces groups and roles in the access control model, providing rules to express policies beyond a single access control policy. Authorizations are expressed by using a Prolog like language. However, this language is not standard and it requires the user have specific mathematical background. More recently, Nicodemos et al. have defined Ponder [26], a declarative language to specify security and management policies for distributed systems. This language specifies different types of policies, grouping them into roles and relationships, and defining configurations of roles and relationships as management structures. Ponder is intended to be extensible and it is also platform-independent. This language shares some characteristics with the largely adopted OCL (Object Constraint Language). Being based on OCL, particular efforts are needed to implement policies, hence making *Ponder* difficult to employ for naive users.

Regarding the task based authorization controls (TBAC) security model [4], the visual language hierarchy (VTBAC) and the Visual Security Administrator (VISA) system implementing it have been defined to simplify the specification and administration of security policies based on TBAC [22]. VTBAC hierarchically combines two visual languages to support security management for non-technical users. The top-tier language is based on the dataflow diagram, which can be customized with metaphors of the specific domain. To specify authorizations and dependencies the second-tier language is used. The VISA system uses a proprietary representation of visual sentences so that it cannot import visual sentences created with other graphical editors. Moreover, access policies cannot be expressed in terms of roles but only in terms of tasks accomplishable by a specific user. Finally, the system does not use a standard policy implementation language like XACML, although it can be adapted to this end thanks to its underlying visual language compiler technology.

A framework for managing security at two distinct layers, separating the policy specification from its implementation is proposed in [23]. The upper layer, specifically conceived for expert users, should contain information on specifying and processing security requirements based on informal statements. On the other hand, the lower layer is a refinement of the upper layer, and contained technical details of security policies. It is worth noting that the approach provides abstract policy specification without mentioning visual assistance mechanisms.

Concerning role-based oriented systems, Wang presents the process support (Chips) system [24], which also integrates visual facilities to specify security policies. The visual environment is used to define permissions across shared workspaces within a hypermedia environment. The workspaces appear on the interface as labelled boxes, which users can edit and navigate through palettes. However, although the system is suitable for non-expert users, it is not customizable for other application domains.

Anderson specifies a profile for the use of the XACML language to meet the requirements of the RBAC model [7]. This specification begins with an explanation of the building blocks from which the RBAC solution is constructed. Then, the specification discusses how these blocks may be used to implement the various elements of the RBAC model. Finally, the normative section of the specification describes compliant uses of the building blocks in implementing an RBAC solution. To model authorization policies based on RBAC an approach based on UML/OCL is proposed in [27]. Similarly to our approach, the authors propose a visual environment, namely USE system (UML Specification Environment), to model access policies. The policy generation is based on the Model Driven Development (MDD) process and it consists of three steps: build the Platform Independent Model (i.e., a model with a high level of abstraction), turn the Platform Independent Model into the Platform Specific Models, and generate access policies from the Platform Specific Models. Unlike this approach, we do not use OCL, hence making the approach more suitable for unexperienced users. In fact, as shown in [29], the use of OCL combined with UML generally offers relevant benefits only after a considerable learning effort.

Koch et al. formalize RBAC by using a graphical specification technique based on a generalization of classical string grammars to nonlinear structures [30]. Access policies are defined and managed by using graph transformation tools. Differently from our approach, policies are not compliant with the XACML standard. An approach based on an extension of RBAC to build secure systems is presented in [31]. In particular, authors define the Model Driven Security (MDS) to specify system models along with their security requirements. The MDS model is implemented in a suite of tools and then integrated in a CASE-tool to automatically generate system architectures, including complete, configured access control infrastructures. To formalize access control requirements, a schema combining different UML modelling languages with a security modelling language is also proposed. These models are used to automatically generate access control infrastructures for server-based applications, built from declarative and programmatic access control mechanisms. Experiments to validate the approach are also presented and discussed. Unlike our approach, the access policies generated by the tool cannot interoperate with different systems and frameworks. Furthermore, authors do not provide any usability study to assess the effectiveness of the proposed UML-based CASE-tool.

Concerning the management of access and security policies based on XACML, Lorch et al. proposed the Globus Toolkit middleware [32]. This middleware integrates a policy management system and a policy enforcement point. This tool also enables the specification and modification of resource policies by administrative parties through a graphical user interface, and the secure association and transport of policies to the policy decision components. However, visual assistance mechanisms are not supported to define and manage access policies.

Regarding the generation of XML based policies, a commercial solution by IBM has also been proposed. In particular, IBM has marketed a policy management workbench, namely the SPARCLE (Server Privacy ARrchitecture and CapabiLity Enablement) system [33]. This system was born as a research prototype to allow policy experts to define or import privacy policy rules in natural language. The tool automatically parses the text to extract the elements of the rules, and enables the expert to review and modify the rules. Then the tool transforms the rules into XML code, hence enabling any enforcement engine to handle access rules.

9. Discussion

Several studies in the literature describe policy management systems and policy enforcement points. These are often integrated into toolkits enabling a security administrator to specify and modify resource policies by using graphical user interfaces. In this paper we have presented several visual languages to support network, system, and platform administrators in the management of resources based on access policies. We have also described a system prototype based upon these languages, also addressing the efforts to support administrators during the design of RBAC policies, and the implementation of XACML policies.

We are currently implementing other policy generators to translate RBAC policies in other standardized policy languages. Thus, by only learning the visual languages, administrators will be able to manage resource access in several standard policy languages. In the future we aim to improve security in the delivery of policies by using emerging standards, in order to allow the secure communication among system components. For instance, to convey the policies from the Visual Policy Editor to the Policy Repository, and to enable the secure communication between PDP and PEP, we are investigating the use of the SAML (Security Assertion Markup Language) standard [34]. We also plan to extend the system network infrastructure to configure and manage both Web based applications and distributed heterogeneous applications, such as ambient intelligence applications, Process Support Systems (PSS), and WorkFlow management systems. Finally, we are currently investigating the possibility of embedding our approach within a J2EE application server in order to facilitate the development of the access control modules in enterprise java based applications. In this way we aim to let programmers focus more on the development of the business logic of an application by relieving him/her from the burden of developing access control modules.

Acknowledgments

We would like to thank Dr. Michele Di Capua and Dr. Sebastiano Manganiello from Unlimited Software s.r.l. who helped us in developing the multimedia CMS and in testing our policy management system and approach on it.

This research has been supported by “Centro Regionale di Competenza - ICT” of Regione Campania, Italy.

References

[1] N.N.C.S. Center, A guide to understanding discretionary access control in trusted systems, Technical Report, United States of America National Security Agency, 1987. URL 〈http://www.radium.ncsc.mil/tpep/library/rainbow/NCSC-TG-003.html〉.
[2] D. Bell, L. LaPadula, Secure computer system unified exposition and multics interpretation, Technical Report MTR-2997, MITRE Corp., Bedford, MA, July 1975.
[3] B. Lampson, Protection, ACM Operating Systems Review 8 (1) (1974) 18–24.
[4] R. Thomas, R. Sandhu, Task-based authorization: a paradigm for flexible and tailorable access control in distributed applications. URL 〈citeseer.ist.psu.edu/article/thomas93taskbased.html〉.
[5] D.F. Ferraiolo, J.F. Barkley, D.R. Kuhn, A role-based access control model and reference implementation within a corporate intranet, ACM Transactions on Information and System Security 2 (1) (1999) 34–64 URL 〈citeseer.ist.psu.edu/ferraiolo99role.html〉.
[6] D.F. Ferraiolo, R.S. Sandhu, S.I. Gavrila, D.R. Kuhn, R. Chandramouli, Proposed NIST standard for role-based access control, Information and System Security 4 (3) (2001) 224–274 URL 〈citeseer.ist.psu.edu/ferraiolo01proposed.html〉.
[7] A. Anderson, Xacml profile for role based access control (RBAC), 2004. URL 〈http://docs.oasis-open.org/xacml/cd-xacml-rbac-profile-01.pdf〉.
[8] O.O. for the Advancement of Structured Information, extensible access control markup language (XACML) version 1.1, 2003. URL 〈http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf〉.
[9] M. Burnett, Visual language research bibliography, 2003. URL 〈http://www.cs.orst.edu/ ~ burnett/vpl.html〉.
[10] F. Ferrucci, G. Tortora, G. Vitiello, Visual programming. In: Encyclopaedia of Software Engineering, Wiley, New York, 2002.
[11] G. Costagliola, G. Polese, Extended positional grammars, in: VL ’00: Proceedings of the 2000 IEEE International Symposium on Visual Languages (VL’00), IEEE Computer Society, Washington, DC, USA, 2000, p. 103.
[12] O.O.M. GROUP, Omg unified modeling language specification, 2003. URL 〈http://www.omg.org/docs/formal/03-03-01.pdf〉.
[13] S.S.V. Graphics, Full 1.2 specification w3c working draft, 2005. URL 〈http://www.w3.org/TR/SVG12/〉.
[14] Apple, Darwin streaming server, 2005. URL 〈http://developer.apple.com/darwin/projects/streaming/〉.
[15] J. Nielsen, Usability Engineering, Academic Press, New York, USA, 1993.
[16] A.N. Oppenheim, Questionnaire Design, Interviewing, and Attitude Measurement, new ed., Martin’s Press, London, 1992.
[17] A.F. Blackwell, Metacognitive theories of visual programming: What do we think we are doing?, in: VL ’96: Proceedings of the 1996 IEEE Symposium on Visual Languages, IEEE Computer Society, Washington, DC, USA, 1996, pp. 240–246.

[18] A.F. Blackwell, T.R.G. Green, Does metaphor increase visual language usability? in: Visual Languages, 1999, pp. 246–253. URL ⟨citeseer.ist.psu.edu/blackwell99does.html⟩.

[19] R.W. Reeder, L. Bauer, L.F. Cranor, M.K. Reiter, K. Bacon, K. How, H. Strong, Expandable grids for visualizing and authoring computer security policies, in: M. Czerwinski, A.M. Lund, D.S. Tan (Eds.), CHI, ACM, New York, 2008, pp. 1473–1482 URL ⟨http://dblp.uni-trier.de/db/conf/chi/chi2008.html#ReederBCRBHS08⟩.

[20] A. Heydon, M. Maimone, J. Tygar, J. Wing, A. Zaremski, Miro: Visual specification of security, IEEE Transactions on Software Engineering 16 (10) (1990) 1185–1197 URL ⟨http://doi.ieeecomputersociety.org/10.1109/32.60298⟩.

[21] E. Saul, A. Hutchison, Team-and-role-based organizational context and access control for cooperative hypermedia environments, in: Proceedings of Annual South African Telecommunication, Networks, and Applications Configuration, 1999, pp. 171–177.

[22] S.K. Chang, G. Polese, M. Cibelli, R. Thomas, Visual authorization modeling in e-commerce applications, IEEE MultiMedia 10 (1) (2003) 44–54 URL ⟨http://doi.ieeecomputersociety.org/10.1109/MMUL.2003.1167921⟩.

[23] J. Leiwo, Y. Zheng, A framework for the management of information security, in: ISW '97: Proceedings of the First International Workshop on Information Security, Springer, London, UK, 1998, pp. 232–245.

[24] W. Wang, Team-and-role-based organizational context and access control for cooperative hypermedia environments, in: HYPERTEXT '99: Proceedings of the Tenth ACM Conference on Hypertext and Hypermedia: Returning to our Diverse Roots, ACM Press, New York, NY, USA, 1999, pp. 37–46 URL ⟨http://doi.acm.org/10.1145/294469.294480⟩.

[25] S. Jajodia, P. Samarati, V.S. Subrahmanian, A logical language for expressing authorizations, sp 00, 1997, p. 0031. URL ⟨http://doi.ieeecomputersociety.org/10.1109/SECPRI.1997.601312⟩.

[26] N. Damianou, N. Dulay, E. Lupu, M. Sloman, Ponder: a language for specifying security and management policies for distributed systems, 2000. URL ⟨http://www-dse.doc.ic.ac.uk/policies/ponder.html⟩.

[27] K. Sohr, G.-J. Ahn, L. Migge, Articulating and enforcing authorisation policies with UML and OCL, SIGSOFT Software Engineering Notes 30 (4) (2005) 1–7 ⟨doi:http://doi.acm.org/10.1145/1082983.1083215⟩.

[28] K.N.L. James A. Hoagland, Raju Pandey, Security policy specification using a graphical approach., Technical Report, Department of Computer Science, University of California, 1998. URL ⟨http://www.radium.ncsc.mil/tpep/library/rainbow/NCSC-TG-003.html⟩.

[29] L.C. Briand, Y. Labiche, H.-D. Yan, M.D. Pent, A controlled experiment on the impact of the object constraint language in uml-based development, in: ICSM '04: Proceedings of the 20th IEEE International Conference on Software Maintenance, IEEE Computer Society, Washington, DC, USA, 2004, pp. 380–389.

[30] M. Koch, L.V. Mancini, F. Parisi-Presicce, A graph-based formalism for RBAC, ACM Transactions on Information and System Security 5 (3) (2002) 332–365 doi: ⟨http://doi.acm.org/10.1145/545186.545191⟩.

[31] D. Basin, J. Doser, T. Lodderstedt, Model driven security: from uml models to access control infrastructures, ACM Transactions on Software Engineering and Methodology 15 (1) (2006) 39–91 doi: ⟨http://doi.acm.org/10.1145/1125808.1125810⟩.

[32] M. Lorch, D. Kafura, S. Shah, An XACML-based policy management and authorization service for globus resources, in: GRID '03: Proceedings of the Fourth International Workshop on Grid Computing, IEEE Computer Society, Washington, DC, USA, 2003, pp. 208–210.

[33] IBM, Sparcle (server privacy architecture and capability enablement). URL ⟨http://www.zurich.ibm.com/pri/projects/sparcle.html⟩.

[34] O.O. for the Advancement of Structured Information, Security assertion markup language, specification set v2.0, oasis security services tc, 2005. URL ⟨http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os-xsd.zip⟩.