Implementing an Access Control System for SVG Documents

E. Fernández-Medina[1], G. Ruiz[1], and S. De Capitani di Vimercati[2]

[1] Escuela Superior de Informática
Universidad de Castilla-La Mancha
13071, Ciudad Real - Spain
Eduardo.FdezMedina@uclm.es
gruiz@proyectos.inf-cr.uclm.es
[2] Dip. di Tecnologie dell'Informazione
Università degli Studi di Milano
26013 Crema - Italy
decapita@dti.unimi.it

Abstract. In this paper, we present an access control system that can be used for controlling access to SVG documents. The first part of this paper briefly describes the access control model on which the system is based. The second part of this paper presents the design and implementation of the system.

1 Introduction

An increasing amount of multimedia information transmitted over the Internet is in the form of vector image data, encoded by means of new XML-based standards such as the World Wide Web Consortium's *Scalable Vector Graphics* (SVG) [10], which allows for the definition of two dimensional vector graphics (specifically vector graphic shapes, images, and text) for storage and distribution on the Web. The SVG standard can be used in different applications: technical plans, organizational charts and diagrams, as well as medical images used in diagnosis and research, to name a few. In the security area, while controlling access to text-based documents has been the focus of many research activities [9], raster graphic information has been seldom considered, mainly because of its monolithic internal structure. However, XML-based vector images present new and challenging *feature protection* problems, related to fine-grained access control to their internal structure. We have then defined a novel approach to fine-grained feature protection of Scalable Vector Graphics (SVG) data [1]. The proposed model allows to selectively transform SVG graphical data according to the user's profile, thus releasing only the features that the user is entitled to see. In this paper we present the design and implementation of an access control system based on this model. More precisely, in Section 2 we show an overview of SVG. In Section 3, we summarize the access control model for SVG documents presented in [1]. In Section 4, we describe the architecture of the system and the algorithms implementing the enforcement of the access control rules. In Section 5

R. Meersman and Z. Tari (Eds.): OTM Workshops 2003, LNCS 2889, pp. 741–753, 2003.
© Springer-Verlag Berlin Heidelberg 2003

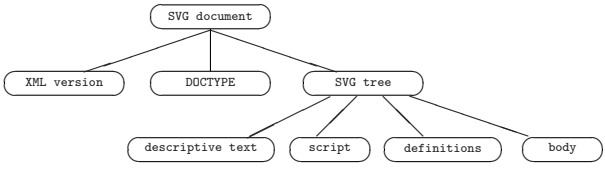


Fig. 1. General Structure of an SVG Document

we illustrate the system's working through an example. Finally, in Section 6 we present our conclusions and future work.

2 Overview of SVG

An SVG document has a structure, composed of several optional elements placed in the document in an arbitrary order. Figure 1 shows the general structure used. Nodes `XML Version` and `DOCTYPE` are common for any XML-based document and specify the XML version used in the document and information about the type of the document (the public identifier and the system identifier for SVG 1.0), respectively. Node `SVG` contains all the elements specific to SVG documents and is composed of four parts: descriptive text, script, definitions, and body. The descriptive text includes textual information not rendered as part of the graphic and is represented by two elements: `<title>`, usually appearing only once, and `<desc>`, appearing several times to describe the content of each SVG fragment. The script portion contains function definitions. Each function is associated with an action that can be executed on SVG objects in the document. Functions have a global scope across the entire document. The definition portion contains global patterns and templates of graphical elements or graphical properties that can be reused in the body of the SVG document. Each definition is characterized by a name, which is used in the body of the document to reference the definition, and by a set of properties. The graphical elements to be rendered are listed after the `<defs>` node, according to the order of rendering. Each element can belong to any of the basic SVG graphics elements, such as `path`, `text`, `rect`, `circle`, `ellipse`, `line`, `polyline`, `polygon`, and `image`, whose names are self-explanatory. The body of an SVG document contains any number of container and graphics elements. A container element can have graphics elements and other container elements as child elements. Container `<g>` is used for *grouping together* related graphics elements. A graphics element can cause graphics to be drawn. For instance, the `use` graphics element references another element (usually a definition) and indicates that the graphical contents of that element must be drawn at that specific point in the document. Each SVG element may have its own properties, modeled by XML attributes. All elements in the document can be uniquely identified including the special attribute `id='identifier'`. It is also possible to include user-defined properties, which can be useful for SVG data processing.

3 An Access Control Model for SVG Documents

The model is based on the use of authorizations (or access control rules) that are themselves expressed with an XML-based language. Each authorization specifies the *subject* to which the authorization applies, the *object* to which the authorization refers, the *action* to which the authorization refers, and the *sign* describing whether the authorization states a permission (sign = '+') or a denial (sign = '−'). Basically, an authorization ⟨*subject*,*object*,*action*,*sign*⟩ states that *subject* is allowed (*sign* = '+') or is denied (*sign* = '−') to perform *action* on *object*.

Subjects. Our model allows authorizations to be referred to specific user identities (e.g., `Sam`), user groups (e.g., `Employee`), and properties of the users (e.g., name, address, and specialty) that are stored in profiles. Each profile is modeled as a semi-structured document and can be referenced by means of XPath expressions [13]. More precisely, the subject field is composed of two parts: (1) the identity of the user/group, and (2) a subject expression, which is an XPath expression on users' profiles. Authorization subjects are then defined as XML elements of the form:

```
<subject>
   <id value='user/group-id'/>
   <subj-expr>xpath-expr</subj-expr>
</subject>
```

Objects. Objects are elements of an SVG document. There are three kinds of protection objects: *definitions* (`<defs>`), *groups* (`<g>`), and *SVG elements*. SVG elements can be graphical or textual elements, such as `rect` or `circle`, or can be elements referencing the definitions. The model allows the association of authorizations with any of such specific elements. Objects can be referenced by using both generic XPath expressions on the SVG document and high level predicates that make the reference independent of the syntax of the elements in the document. More precisely, the model defines four ways of identifying objects:

- a *path expression* resolving in the object;
- an *object identifier* (the value of attribute `id`);
- a *type* (the value of attribute `typeElement`);
- a function `perimeter` that can take as input a path expression, an object identifier, or a type and returns the area that contains the object given as input.

Authorization objects are then defined as XML elements of the form:

```
<object>
   <refer value='object-id'/>
   <cond>pred-expr</cond>
</object>
```

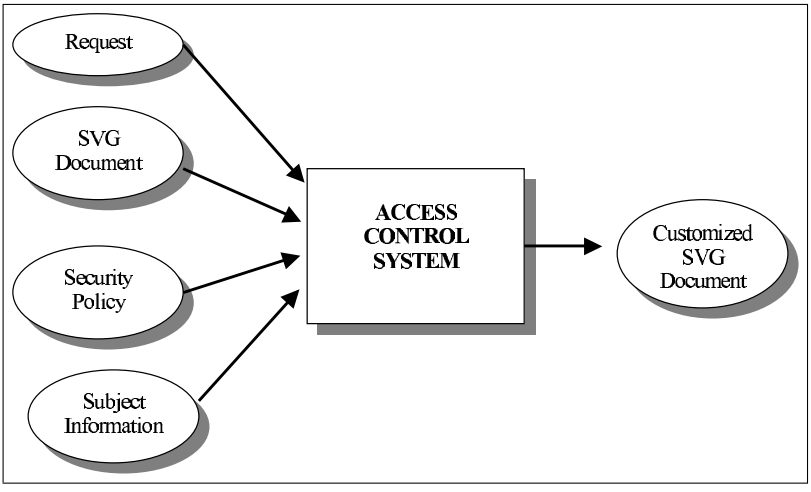


Fig. 2. Hight Level Scheme

This structure allows for specifying an object or a set of objects, and a condition for referencing all elements satisfying specific semantically rich constrains (e.g., computers inside a room). Conditions are boolean expressions that can use the following *predicates*:

- `inside(`*obj*`)`. It returns the object in the authorization if it is inside an element whose identifier, type, or name is *obj*.
- `together_with(`*obj*`)`. It returns the object in the authorization if it is a child of an element together with an object whose identifier, type, or name is *obj*.
- `number_of(`*obj*`,`*n*`)`. It returns the object in the authorization if there are *n* instances of the object whose identifier, type, or name is *obj*.

Action. Actions can be render, rotate, change color, and so on. For the sake of simplicity, we consider as default action the rendering of an SVG document, so in the authorizations we do not explicitly specify an 'action' element.

4 Implementation of the System

The model briefly summarized in this paper has been implemented by using the ASP technology [7] under the Microsoft Internet Information Server [5]. The language used to implement all the algorithms of the system is Visual Basic Script [8]. To manage the XML Document Object Model (DOM) [2], we have selected the parser MSXML [6] that provides a rich set of functions and procedures. Figure 2 illustrates the abstract working of our system. Given an access request, the involved `SVG document`, the `security policy` containing all the authorizations defined on the document, and the `subject information` (i.e.,

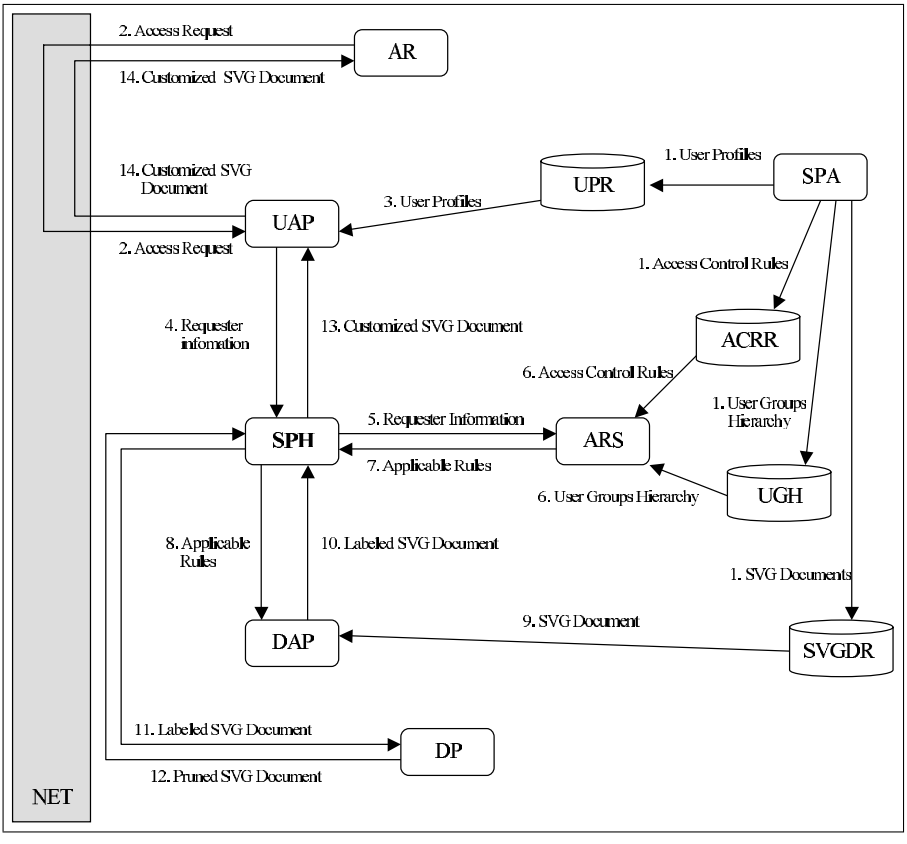


Fig. 3. System Architecture

the user group hierarchy definition and users' profiles), the system has to generate the customized SVG document that includes only the features that the requester is entitled to see.

We now describe the architecture of the system (Section 4.1), and the main algorithms enforcing the security policy (Section 4.2) together with their complexity.

4.1 System Architecture

The architecture of the system (see Fig. 3) is composed of a set of data repositories, a set of modules, and several data flows.

Data repositories. The system includes four basic repositories:

- *User Profiles Repository (UPR)*: It stores the XML-based user profiles.
- *Access Control Rules Repository (ACRR)*: It stores the XML-based documents that contain all the authorizations defined in the security policies.

- *User Groups Hierarchy (UGH)*: It stores the information about the user groups hierarchy.
- *SVG Documents Repository (SVGDR)*: It contains all the SVG documents that have to be protected.

Modules. The modules included in the system are the following.

- *Security Policy Administrator (SPA)*. It creates and manages all the repositories.
- *Access Requester (AR)*. It manages the access request information.
- *User Access Point (UAP)*. It is in charge of managing the user authentication.
- *Security Policy Handler (SPH)*. It coordinates the communication between the modules to fulfill the general goal of the system.
- *Applicable Rules Selector (ARS)*. It considers all the access control rules specified in the security policy, and selects only those that are applicable to the requester.
  q
- *Document Analyzing Point (DAP)*. It processes the SVG document, propagating permission and denial labels in the SVG tree according to the authorizations selected by the ARS.
- *Document Pruner (DP)*. It prunes a labeled SVG document, leaving only the SVG features that the requester is entitled to see.

Enforcement. The client-server model enforces the security policy as follows.

1. The SPA creates all repositories (UPR, ACRR, UGH and SVGDR) in the server.
2. The AR sends the access request and the login and password of the requester to the UAP.
3. The UAP reads from the UPR module the requester's profile.
4. The UAP prepares an XML request document that includes the requester's profile and information about the document the requester wants to access. This XML document is sent to the SPH.
5. The SPH coordinates the other modules. It initially sends the requester information to ARS.
6. The ARS reads from the ACRR the access control rules, and from the UGH the user groups hierarchy.
7. Once the ARS has collected the information about the requester, the access control rules, and the user groups hierarchy, it selects the access control rules that are applicable to this particular requester, according to her profile and to the user group membership. The set of applicable access control rules are then sent to the SPH.
8. The SPH sends the applicable access control rules to the DAP.
9. The DAP reads from the SVGDR the SVG document.

10. Once the DAP has the applicable rules and the SVG documents, it starts the labeling process on the SVG document. Each rule is processed, and the corresponding features of the SVG document are labeled. At the end of this process, the labeled document is sent to the SHP.
11. The SPH sends the labeled SVG document to the DP in order to obtain the visible document.
12. The DP prunes all the SVG elements that are denied and sends the resulting document to the SPH.
13. The SPH sends the customized document to the UAP.
14. The UAP offers the rendering of the resulting SVG document.

4.2 System Development

As we have described in the previous section, we can identify four main processes: user information gathering, applicable rules selection, SVG document labeling, and SVG document pruning. In the following, we give some implementation details of these processes.

User Information Gathering Algorithm. The UAP module is in charge of managing the user authentication. To this end, the algorithm needs to access the authentication information in the XML document that contains the user profiles. The corresponding XPath query is:

```
/users/user-profile[login[@value='" + login + "'] and
          password[@value='"+password+"']]
```

Once we have identified the requester, we just have to create a XML document that includes the requester information and the object that she wants to access.

Complexity. Let `n` be the number of user profiles stored in the system. The complexity of this algorithm is $\Theta(\mathrm{n})$.

Applicable Rules Selection Algorithm. The security policy can include many access control rules and the system has to determine the rules that applies to the requester. To this purpose, the requester information is sent from the SPH to the ARS, which is in charge of reading all the access control rules from the ACRR and the user groups hierarchy from the UGH, and then to filter only those rules that involve the requester. The algorithm is as follows:

1. For each access control rule, we identify all affected users and decides if the requester is one of them. We first consider the `subject id`:
   - If the subject id corresponds to an individual user `u` and `u` is the requester, then the rule is considered, otherwise it is discarded.
   - If the subject id corresponds to a group `g`, and the requester is a direct or indirect member of `g`, the rule is considered, otherwise it is discarded.

The rules thus selected have to be taken into consideration if and only if the requester's profile satisfies the XPath expression that can be included in such rules.

2. The applicable rules are ordinated according to their priority level, from the lowest to the the highest. More precisely, the priority level is established by comparing the `subject id` of the rules: rule `r` has a priority greater than rule `r′` if the `subject id` of `r` is an ascendant of the `subject id` of `r′` in the user group hierarchy. Incomparable rules or with the same level of generality or specificity have the same priority. In case of conflict rules (i.e., a positive and a negative rule both applicable) that are incomparable or with the same level of generality or specificity, the denials take precedence principle is applied. This means that the priority of a negative rule is greater than the priority of a positive rule.

At the end of this process, we obtain an ordered list of applicable rules that has to be sent to the SPH.

Complexity. Let `n` be the number of access control rules and `m` be the number of groups. The complexity of point 1 of the algorithm is Θ(n*lg m), where lg m is derived from the binary search of the user groups tree. Point 2 of the algorithm has been implemented by applying the `quick sort` algorithm and therefore the complexity is Θ(n*lg n*lg m). So, we can conclude that the complexity of the complete algorithm is Θ(n*lg n*lg m).

SVG Document Labeling Algorithm. The labeling process consists in associating a positive or negative label with each element in the SVG document. Once the SPH sends the ordered list of applicable access control rules to the DAP, the SVG document has to be read from the SVGDR, and all these rules has to be processed in order to decide which elements of the document the requester is entitled to see.

Positive rules (sign='+') generates positive labels associated with the corresponding SVG nodes, and negative rules (sign='−') generates negative labels associated with the corresponding SVG nodes.

The labeling process is basically composed of two steps:

1. *Top-down labeling*. For each rule in the ordered list the following process is applied:
   - Determine the SVG elements that satisfy the object specification.
   - A label with the sign of the rule is associate with each selected SVG element. If some of these elements have already a sign label, it is overwritten, because previous labels corresponds to more general rules.
   - Labels are then recursively propagated to all SVG elements that are contained in the affected SVG elements.
2. *Bottom-up labeling*. Whenever a node has a positive label and the corresponding father has a negative label, the labeling process has to go up in the SVG tree, making visible the perimeters of all the containers. This process

is realized by applying a depth-first search: for each SVG node `n`, if the sign of `n` is '+' the sign of the perimeter node of all ascendant of `n` is set to '+'.

The resulting labeled SVG document is then sent to the SPH module to prepare the definitive document.

Complexity. Let `n` be the number of applicable rules and `m` be the number of SVG nodes (elements that compose the tree of the SVG document). The complexity of the top-down and bottom-up labeling processes is Θ(n*m) and Θ(m), respectively.

SVG Document Pruning Algorithm. The pruning process consists in deleting all nodes of the labeled SVG document that the requester is not entitled to see. In particular, if the security policy is open (i.e., an element is accessible unless a negative rule specifies the opposite), we have to delete all nodes with a negative label. On the other hand, if the security policy is close (i.e., an element cannot be accessed unless a positive rule specifies the opposite), we have to delete all non-labeled nodes and all nodes with a negative label. We consider the open security policy as default. The pruning process is carried out by the DP module, and works as follows:

- The SVG tree is traversed by means of a depth-first search, and for each node:
  - If the node is a simple node (not a container), and its label is '−', we prune the node.
  - If the node is a container and its label is '−', we have to prune the node, but only if all its children have also a negative '−' label. Otherwise, this node remain intact, and the children are processed.

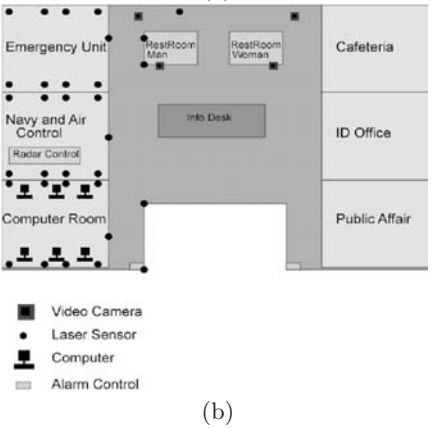
Complexity. Let `n` be the number of nodes in the SVG tree. The complexity of this algorithm is Θ(n).

5 An Example

To illustrate the working of the system, consider the simple SVG document in Fig. 4a rendered in Fig. 4b. It represents the map of a Department of Defense (DD). The body of the SVG document is a group element whose identifier is `deptdefense`, and its sub-elements are `main_entrance`, `public area`, and `private_area`. Group `public_area` includes `cafeteria`, `IDoffice`, two `restroom` instances, `info_desk`, and `public_affair` office. Group `private_area` includes `emerg_unit`, `navy and air control`, `computer_room`, four `videocamera` instances, eight `laser_sensor` instances, and two `alarm_control` instances. Each of these elements are composed of a graphical representation and a `name`. Consider also the user groups hierarchy in Fig. 5a. Here, the root element `users` has two subgroups: `Non DD Members` and `DD`

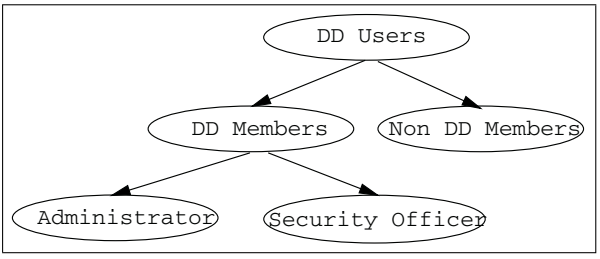
```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" .....
<svg width="600" height="600">
   <defs>
      <rect id="rectRoom" width="150" height="132" style="stroke:rgb(0,0,0);fill:rgb(255,255,0)"/>
      <symbol id="computer" viewBox="0 0 25 25">.....</symbol>
      .....
   </defs>
   <g id="deptDefense">
      <g id="perimeterdeptDefense" perimeter="yes" typeElement="Perimeter">
         <polygon ...../>
      </g>
      <g id="PublicArea">
         <g id="Cafeteria">
            <g id="perimeterCafeteria" perimeter="yes" typeElement="Perimeter">
               <use x="450" y="0" xlink:href="#rectRoom"/>
            </g>
            <text x="470" y="70" style="font-size:20"> Cafeteria </text>
         </g>
         .....
      </g>
      <g id="PrivateArea">
         <g id="EmergencyUnit">
            <g id="perimeterEmergencyUnit" perimeter="yes" typeElement="Perimeter">
               <use x="0" y="0" xlink:href="#rectRoom"/>
            </g>
            <text x="5" y="70" style="font-size:20"> Emergency Unit </text>
            <g id="LaserSensonrsEU">
               <use x="10" y="10" xlink:href="#LaserSensor" typeElement="security"/>
               .....
            </g>
         </g>
         <g id="NAControl">
            <g id="perimeterNAControl" perimeter="yes" typeElement="Perimeter">
               <use x="0" y="132" xlink:href="#rectRoom"/>
            </g>
            <text x="5" y="180" style="font-size:20"> Navy and Air</text>
            <text x="20" y="200" style="font-size:20"> Control</text>
            <g id="radarControl">
               <rect id="radarControlrect" ...../>
               <text x="15" y="230" style="font-size:15"> Radar Control</text>
            </g>
            <g id="LaserSensonrsNAC">
               <use x="10" y="140" xlink:href="#LaserSensor" typeElement="security"/>
               .....
            </g>
         </g>
      </g>
         <g id="ComputerRoom">
            <g id="perimeterComputerRoom" perimeter="yes" typeElement="Perimeter">
               <use x="0" y="265" xlink:href="#rectRoom"/>
            </g>
            <text x="0" y="330" style="font-size:20">Computer Room</text>
            <g id="computers">
               <use x="20" y="270" width="30" height="30" xlink:href="#computer" typeElement="computer"/>
               .....
            </g>
            <g id="LaserSensonrsCR">
               <use x="10" y="270" xlink:href="#LaserSensor" typeElement="security"/>
               .....
</svg>
```

(a)



(b)

Fig. 4. An example of SVG document (a) and its rendering (b)




```
<subjects>
  <user-profile id="Alice">
    <name value="AliceDoo"/>
    <group value="SecurityOfficer"/>
    <address value="University Dr."/>
    <job value= "head"/>
    <level value="senior"/>
    <login value="AliceD"/>
    <password value="Alice133"/>
  </user-profile>
  <user-profile id="Bob">
    <name value="Bob"/>
    <group value="Administrator"/>
    <address value="Manchester Rd"/>
    <citizenship value="EU"/>
    <job value= "doctor"/>
    <login value="BobR"/>
    <password value="BobbyRT"/>
  </user-profile>
</subjects>
```

(a) (b)

Fig. 5. An example of user group hierarchy (a) and two examples of user profiles (b)

`Members` that in turn has two subgroups, namely `Administrator` and `Security Officer`. Figure 5b shows the profile of two users, `Alice` and `Bob`. These profiles show among other details, that `Alice` is a member of the `Security Officer` group and `Bob` is a member of the `Administrator` group.

Suppose now that the access to the given SVG document has to be regulated according to the following three authorizations.

1. Everybody can see the content of any room in the public area.

```
<authorization-rule>
  <subject>
    <id value="users"/>
  </subject>
  <object>
    <refer value = "id.PublicArea"/>
  </object>
  <sign>+</sign>
</authorization-rule>
```

2. Members of the `Security Officer` group whose job is not `controller` cannot see computers.

```
<authorization-rule>
  <subject>
    <id value="SecurityOfficer"/>
    <subj-expression>
      /subjects/user-profile[job[not(@value='controller')]]
    </subj-expression>
  </subject>
  <object>
    <refer value = "typeElement.computer"/>
  </object>
  <sign>-</sign>
</authorization-rule>
```

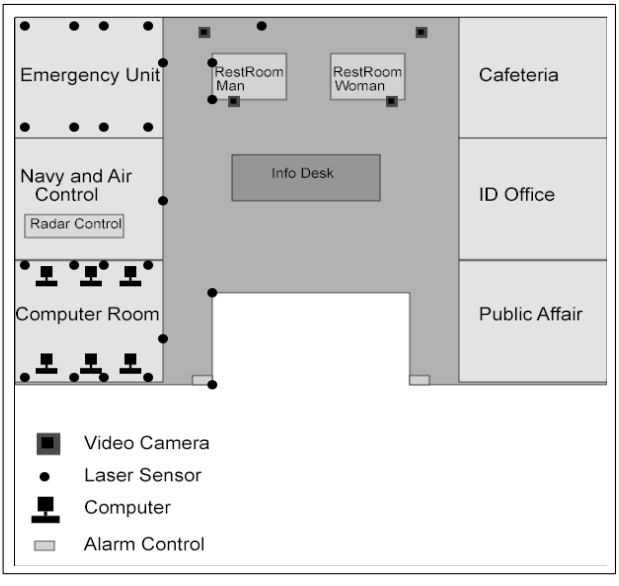


Fig. 6. Final SVG rendering

3. Members of the `Administrator` group cannot see the security elements in the `navy and air control` room.

```
<authorization-rule>
  <subject>
    <id value="Administrator"/>
  </subject>
  <object>
    <refer value = "typeElement.security"/>
    <cond>inside(id.NAControl)</cond>
  </object>
  <sign>-</sign>
</authorization-rule>
```

Consider now a request to read the map of the department submitted by user `Bob`. The `applicable rules selection algorithm` selects the first rule and the third rule; the second rule does not match the profile of `Bob`. Once the applicable rules have been selected, the `labeling algorithm` includes the authorization labels in the SVG document, and finally, the `pruning algorithm` eliminates all the SVG documents that `Bob` is not entitled to see. Figure 6 illustrates the portion of the map returned to `Bob`. As you can notice, according to the first rule, `Bob` is entitled to see the public area, and, according to the third rule, `Bob` cannot see all the security elements included in the `navy and air control` room.

6 Conclusions and Future Work

We have presented the design and implementation of an access control system for SVG documents. We have analyzed the architecture, and the algorithms used in the system. The system allows for controlling access to any SVG document in a user-transparent way and is currently used (in a experimental way) for controlling access to a graphical representation of the building of a computer science faculty.

Future work includes the implementation of an administrative tool for managing all the system components (access control rules, users, groups, and so on). Another important aspect is the extension of the model to control access to other XML-based multimedia standards formats, such as SMIL [11] for multimedia presentations, VoiceXML [12] for dialog, and MPEG-21 [3] and MPEG-7 [4] for video.

References

1. E. Damiani, S. De Capitani di Vimercati, E. Fernandez-Medina, and P. Samarati. An access control system for svg documents. In *Proc. of the Sixteenth Annual IFIP WG 11.3 Working Conference on Data and Application Security*, University of Cambridge, UK, July 2002.
2. Philippe Le Hégaret. *DOM*. W3 Consortium, June 2002. http://www.w3.org/DOM/.
3. International Organisation for Standardisation. *MPEG-21 Overview v.5*, October 2002. http://www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm.
4. International Organisation for Standardisation. *MPEG-7 Overview v.8*, July 2002. http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm.
5. Microsoft Corporation. *Internet Information Server*. www.microsoft.com/iis.
6. Microsoft Corporation. *MSXML 3.0 SDK*. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk30/htm/xmmscxmloverview.asp.
7. Microsoft Corporation. *Active Server Pages*, December 2000. http://msdn.microsoft.com/library/default.asp?url=/nhp/default.asp?contentid=28000522.
8. Microsoft Corporation. *VB Script fundamentals*, 2003. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/vbstutor.asp.
9. P. Samarati and S. De Capitani di Vimercati. Access control: Policies, models, and mechanisms. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, LNCS 2171. Springer-Verlag, 2001.
10. World Wide Web Consortium. *Scalable Vector Graphics (SVG) 1.0 Specification*, September 2001.
11. World Wide Web Consortium. *Synchronized Multimedia Integration Language (SMIL 2.0)*, August 2001. http://www.w3.org/TR/smil20.
12. World Wide Web Consortium. *Voice Extensible Markup Language (VoiceXML) Version 2.0*, April 2002. http://www.w3.org/TR/voicexml20.
13. World Wide Web Consortium (W3C). *XML Path Language (XPath) 2.0*, December 2001. http://www.w3.org/TR/xpath20.