AN ACCESS CONTROL SYSTEM FOR SVG DOCUMENTS

E. Damiani[1], S. De Capitani di Vimercati[2], E. Fernández-Medina[3], and P. Samarati[1]

(1) Dipartimento di Tecnologie dell'Informazione - Università di Milano, Crema, Italy
{damiani,samarati}@dti.unimi.it

(2) Dipartimento di Elettronica per l'Automazione - Università di Brescia, Brescia, Italy
decapita@ing.unibs.it

(3) Escuela Superior de Informática - Univ. of Castilla-La Mancha, Ciudad Real, Spain
Eduardo.FdezMedina@uclm.es

Abstract The monolithic nature of traditional raster images makes controlled dissemination of their internal features a difficult task. Recently, however, XML-based graphics formats such as the Scalable Vector Graphics (SVG) standard are becoming increasingly popular due to their recognized advantages in terms of application interoperability. In this paper we exploit the XML-based data model of SVG to present a model and a syntax aimed at selectively controlling access to graphic information on the Internet.

Keywords: Access Control, SVG Documents, Vector graphic

1. INTRODUCTION

Vector graphics is a time-honored technique that uses geometrical formulas to represent images, achieving more flexibility than usual raster graphics relying on bit maps. For instance, vector-oriented images can be resized and stretched without any loss of image quality; also, repetitive geometric elements can be defined once and used many times, so that high-quality vector images often require less memory than lower quality bit-mapped ones. While in the past vector graphics was confined to computationally intensive design applications, it is now spreading to new application fields. An increasing amount of the multimedia information being 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) [6], which allows describing two dimensional vector graphics (specifically vector graphic shapes, images, and text) for storage and distribution on the Web. In contrast to raster image format such as GIF, JPEG, and PNG, SVG has many advantages:

- SVG documents are plain text, so they can be read and modified easily.
- Being SVG a vector format, SVG images can be resized without loss of quality and printed at any resolution. Also, graphical objects can be easily grouped, restyled, and transformed.
- Sophisticated interactive and dynamic applications of SVG are made possible by the *Document Object Model* (DOM) [7] underlying all XML-based formats. User interaction is managed via a rich set of *event handlers* that can be assigned to any SVG graphical object.
- SVG offers all the advantages of XML, including interoperability, internationalization (via its support of the Unicode character sets), XSLT restructuring capability [8], and easy manipulation through standard DOM APIs.

The current trend toward XML-based vector graphics is affecting different types of data, such as technical plans, organizational charts and diagrams, as well as medical images used in diagnosis and research. While controlling access to text-based documents has since long been a focus of research activities [5], raster graphic information has been seldom processed with much concern for access control, mainly because of its monolithic internal structure: either a user is allowed to see a bitmap image, or she is not. On the other hand, XML-based vector images present new and challenging *feature protection* problems, related to fine-grained access control to their internal structure. Of course, the feature protection problem could also be solved by storing graphical data in multiple copies at different levels of detail but this solution is seldom practical. For instance, in a hospital, if some MRI-scan images are to be released for research purposes, they must be duplicated omitting any identifying information, making their distribution slow and costly [10, 11]. In this paper, we present a novel approach to fine-grained feature protection of SVG data. Our approach allows to selectively transform SVG graphical data according to the user's profile, releasing only the features that the user is entitled to see. While leveraging on our proposal for protecting XML sources [3], the approach presented in this paper exploits the peculiar characteristics of SVG documents and provides a simple, yet expressive, solution for specifying authorization subjects.

2. A CONCISE OVERVIEW OF SVG

An SVG document has a flexible structure, composed of several optional elements placed in the document in an arbitrary order. After the specification of the XML version used in the document and information about the type of the document, there is node `SVG` that 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 (e.g., `path`, `text`, `rect`) 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 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 attributes. All elements in the document can be uniquely identified including the special attribute `id`. It is also possible to include user-defined properties, which can be useful for SVG data processing.

2.1. Running Example

Figure 1 illustrates the rendering of a sample SVG document, showing the oncology floor of a hospital, which will be used as a running example throughout the paper. The document, integrated in a web site, allows the hospital staff to know both details of the floor (e.g., rooms and equipments location) and recovered patient information. In particular, the rectangular appearing at the bottom with the text provides the information of the patient of bed `1B` on which the mouse is currently positioned (moving the mouse on other beds the corresponding patient will be returned). Figure 2(a) shows a tree-based representation of the document rendered in Figure 1, reporting the types associated

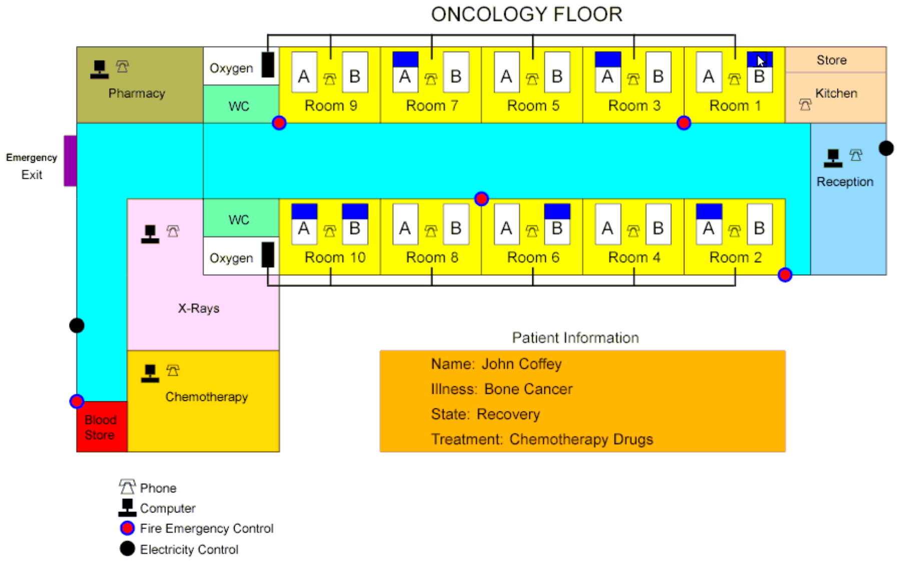
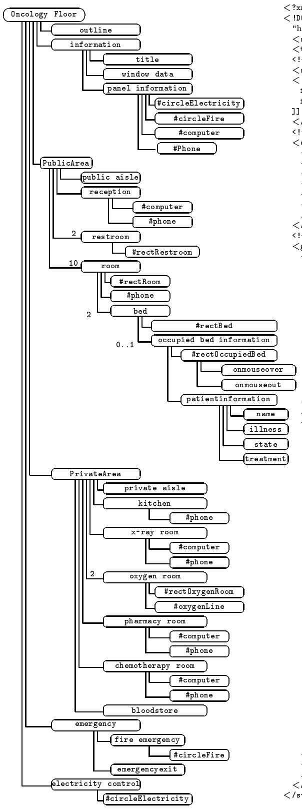


Figure 1. An example of graphic corresponding to an SVG document

with the group elements composing its body. In particular, the body is a group element with `oncologyfloor` as identifier and with sub-elements of type `outline`, `information`, `PublicArea`, `PrivateArea`, `emergency`, and `electricity control` (the document defines one group for each of them). Group `PublicArea` includes `public aisle`, `reception`, two `restroom` instances, and ten `room` instances. Each room, in turn, is composed of a graphical representation (`rectRoom` definition), a name, and two `beds`. Each bed is composed of a graphical representation (`rectBed` definition) and a name. Occupied beds further include a group with a new graphic element (`rectOccupiedBed` definition) and information on the occupying patient. The graphical representation of an occupied bed has two event handlers (`onmouseover=‘display_information(evt)’` and `onmouseout=‘hide_information(evt)’`), which show and hide respectively the patient information as the mouse pointer is positioned over the bed or moved out. Figure 2(b) gives a portion of the corresponding SVG document.

3. THE FEATURE PROTECTION MODEL

Our approach is based on the use of authorization rules that are themselves expressed with an XML-based language. Each authorization rule specifies the *subject* to which the rule applies, the *object* to which the rule refers, the *action* to which the rule refers, and the *sign* describing



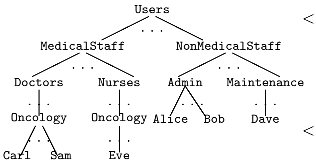
(a)

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
 "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
 <svg width="24cm" height="16cm" viewBox="900 400 3600 2200">
 <title> Small Fragment of a Oncology Floor </title>
 <!-- SCRIPT portion starts here -->
 <script type="text/ecmascript">
 <![CDATA[
  function display_information(evt) { ......... }
  function hide_information(evt) { ......... }
 ]]>
 </script>
 <!-- DEFINITION portion starts here -->
 <defs>
   <rect id="rectRoom" width="400" height="300" stroke="black" fill="beige"/>
   <rect id="rectBed" width="100" height="160" stroke="black" fill="white"/>
   <rect id="rectOccupiedBed" width="100" height="60" stroke="black" fill="blue"/>
   <symbol id="computer" viewBox="0 0 20 20"> ...... </symbol>
   <symbol id="phone" viewBox="0 0 20 20"> ...... </symbol>
   <linearGradient id="MyGradient"> ...... </linearGradient>
   .........
 </defs>
 <!-- BODY portion starts here -->
 <g id="oncologyfloor">
   <g id="information">
     <g id="title">
       <text x="1920" y="700" font-size="80"> ONCOLOGY FLOOR </text>
     </g>
     <g id="window-data">
       <rect id="data" fill="url(#MyGradient)" x="1500" y="1500"
        width="1600" height="400" stroke="black" />
       <text x="2020" y="1470" font-size="60"> Patient Information </text>
       <text x="1700" y="1570" font-size="60"> Name: </text>
       <text x="1700" y="1670" font-size="60"> Illness: </text>
       <text x="1700" y="1770" font-size="60"> State: </text>
       <text x="1700" y="1870" font-size="60"> Treatment: </text>
     </g>
     .........
   </g>
   .........
   <g id="room1" typeElement="room" >
     <use x="2700" y="300" xlink:href="#rectRoom" />
     <g typeElement='content'>
       <text x="2800" y="550" font-size="60"> Room 1 </text>
       <use x="2870" y="400" width="60" height="60" xlink:href="#phone" />
       <g id="A" typeElement="bed" >
         <use x="2750" y="320" xlink:href="#rectBed" />
         <text x="2775" y="440" font-size="70" > A </text>
       </g>
       <g id="B" typeElement="bed" >
         <use id="1B" x="2950" y="320" xlink:href="#rectBed" />
         <text x="2975" y="440" font-size="70" > B </text>
         <g id="bed1B" typeElement="occupiedBedInforamtion">
           <use id="1B" x="2950" y="320" xlink:href="#rectOccupiedBed"
           onmouseover="display_information(evt)" onmouseout="hide_information(evt)"/>
           <g id="patientBed1B" typeElement="infoPatient" visibility="hidden">
             <text typeElement="name" x="1900" y="1570" font-size="60">
               John Coffey
             </text>
             <text typeElement="illness" x="1910" y="1670" font-size="60">
               Bone Cancer
             </text>
             <text typeElement="state" x="1880" y="1770" font-size="60">
               Recovery
             </text>
             <text typeElement="treatment" x="2010" y="1870" font-size="60">
               Chemotherapy Drugs
             </text>
           </g>
         </g>
       </g>
     </g>
   </g>
   <g id="room2" typeElement="room" > ......... </g>
   .........
 </g>
</svg>
```

(b)

Figure 2. Tree-based graphical representation (a) of an SVG document (b)

whether the rule states a permission (sign = '+') or a denial (sign = '−') for the access. Here, for simplicity, we assume action to be the request to render the document (intuitively the `read` operation). This



(a)

```
<user_profile id=‘Dave’>
  <name value=‘Dave’/>
  <address value=‘Forest Ave.’/>
  <job value=‘maintenance worker’/>
  <level value=‘senior’/>
  <building value=‘ADM125’/>
  <office value=‘33’/>
</user_profile>
```

```
<user_profile id=‘Sam’>
  <name value=‘Sam’/>
  <address value=‘Manchester Rd’/>
  <citizenship value=‘EU’/>
  <job value=‘doctor’/>
  <specialization value=‘oncology’/>
</user_profile>
```

(b)

Figure 3. An example of user group hierarchy (a) and user profiles (b)

is not limiting as reference to specific actions defined on the document (e.g., `rotate`) can be regulated by allowing (or not allowing) access to the corresponding element. Given this, we can now focus on subjects and objects of our rules.

3.1. Authorization Subjects

The specification of subjects has often two apparently contrasting requirements [5]. On the one side, subject reference must be simple, to allow for efficient access control and for exploiting possible relationships between subjects in resolving conflicts between the authorizations (e.g., most specific relationships between groups and subgroups). On the other side, one would like to see more expressiveness than the simple reference to user identities and groups, providing support of profile-dependent authorizations whose validity depend on properties associated with users (e.g., age, citizenships, or field-of-specialization) [1, 2]. Our solution nicely encounters both requirements by supporting both user groups and user profiles. As usual, *groups* are sets of users hierarchically organized: groups can be nested and need not be disjoint [4] (e.g., see Figure 3(a)). In addition, for each user, a profile may be maintained specifying the values of properties that our model can exploit to characterize them. The profile is modeled as a semi-structured document and can then be referenced by means of path expressions [9]. A *path expression* is a sequence of element names or predefined functions separated by character / (slash) and is used to identify the elements and attributes within an XML-based document. For instance, with respect to the profiles in Figure 3(b), path expression `user_profile//[./citizenship[@value = ‘EU’]` AND `[./job[@value = ‘doctor’]]` returns element `user_profile` of profiles of EU citizens who work as doctors, that is, it would return the profile of user `Sam`. As illustrated in Figure 4(a), the subject component of our authorization rules then includes an *identity*, whose value

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

(a)

```
<subject>
  <id value='MedicalStaff'/>
  <subj-expr>user_profile//[./citizenship/@value='EU']</subj-expr>
</subject>
```

(b)

Figure 4. The subject component (a) and an example (b)

can be a user or a group identifier, and a *subject expression* which is an path expression on users' profiles. For instance, the `subject` element in Figure 4(b) denotes European users belonging to group `MedicalStaff`.

Note that the purpose of using path expressions in our context is not to retrieve elements and attributes satisfying certain criteria, but to determine whether a given profile (that of the requestor) satisfies the criteria. Intuitively, the authorization rule should apply only if the profile of the requestor satisfies the constraints expressed in the path expression. Given this, we ignore the result of a path expression and simply consider whether it is satisfied (meaning its result is not empty) or not.

3.2. Authorization Objects

According to the description in Section 2, we identify three kinds of protection objects: *definitions* (`defs`), *groups* (`g`), and *SVG elements*, such as `rect` or `circle`, or elements referencing the definitions (e.g., element `use` in Figure 2(b)). Our authorization model allows the association of authorizations with any of such specific elements; generic path expressions on the SVG document can be used to specify the elements to which an authorization applies [3]. Although generic path expressions are sufficient to provide fine-grained authorization specification, their only support result limiting from the point of view of the authorization administration. While verbose, these path expressions refer to the syntax and are detached from the semantics of the elements in the SVG document. As a result, the translation of high-level protection requirements into corresponding path expressions on the document is far from being trivial. It is therefore important to provide a higher level support for the specification of authorization objects. Providing higher level support for the definition of authorization objects translates into solving two problems: (1) *object identification*, that is, how to identify the portion (element) of the SVG document to which an authorization refers; and (2) *condition support*, that is, how to specify conditions that the identified element/s have to satisfy.

Object identification. To provide an expressive authorization language, we exploit the free format of SVG documents by assuming that *semantics* aware tags and good design techniques are defined and exploited. In particular, identifiers (attribute `id`) provide useful as they permit explicit reference to specific elements (e.g., `room1`) based on their name. However, identifiers are not sufficient as listing explicit elements may in some situations result inconvenient (e.g., to protect all rooms of the floor we will have to specify one authorization for each room identifier). Also, support for distinguishing the shape of an object from its content seems to be needed (e.g., to support cases where a user can see the existence of a room - and then its shape - but cannot see what is inside the room). We address these two requirements as follows. First, in addition to the identifier, we allow each element to have an attribute `typeElement` that defines the *conceptual type* of the element (e.g., room, bed, telephone, computer). The type element can be exploited to allow reference to all objects of a given type (e.g., all rooms) in a single expression. Second, if the *shape* of an element is conceptually meaningful we assume a good design of the element where the shape (i.e., the *drawing instructions*) appears at the first level and the content appear in a nested element group.[1] Our predefined function `perimeter()` identifies the shape (i.e., the drawing instructions) of an element (referenced via its identifier or type). Summarizing, an object can then be referenced by means of: a *path expression* resolving in the object; its *object identifier* (value of its attribute `id`); its *type* (value of its attribute `typeElement`); or the application of function `perimeter` to any of the above. To distinguish which of these means is adopted, we use a dot notation prefixing the object with either `id.`, `type.`, or `path.`. For instance, value '`type.room`' indicates objects with `typeElement` equal to room while value '`id.room1`' denotes the room with `id`'s value `room1`.

Condition support. To provide a way for referencing all elements satisfying specific semantically rich conditions, we allow the specification of *object conditions* that identify a set of objects satisfying specific properties. For instance, we may need to define an access rule stating that "a doctor can see computers only if they are in the same room as (i.e., *together with*) diagnostic machines". In our model, conditions are boolean expressions that can make use of the following *predicates*:

- `inside(`*obj*`)`. It returns the object in the authorization rule if it is inside an element whose identifier, type, or name is *obj*.

[1] Note that we are not forcing documents to obey this structure: if the structure is obeyed it can be exploited for the specification of authorizations.

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

(a)

```
<object>
   <refer value='type.phone'/>
   <cond>together_with(type.computer)</cond>
</object>
```

(b)

Figure 5. The object component (a) and an example (b)

- `together_with(`*obj*`)`. It returns the object in the authorization rule 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 rule if there are *n* instances of the object whose identifier, type, or name is *obj*.

Authorization objects in our model are then defined as illustrated in Figure 5(a), where element `refer` provides the object identification and element `cond` specifies additional conditions. For instance, the object in Figure 5(b) denotes all 'phones' that are in the same room as (together with) a 'computer'. With respect to our example in Figure 1, it denotes the phones in the 'Pharmacy', 'X-Rays', 'Chemotherapy', 'Kitchen', and 'Reception' rooms.

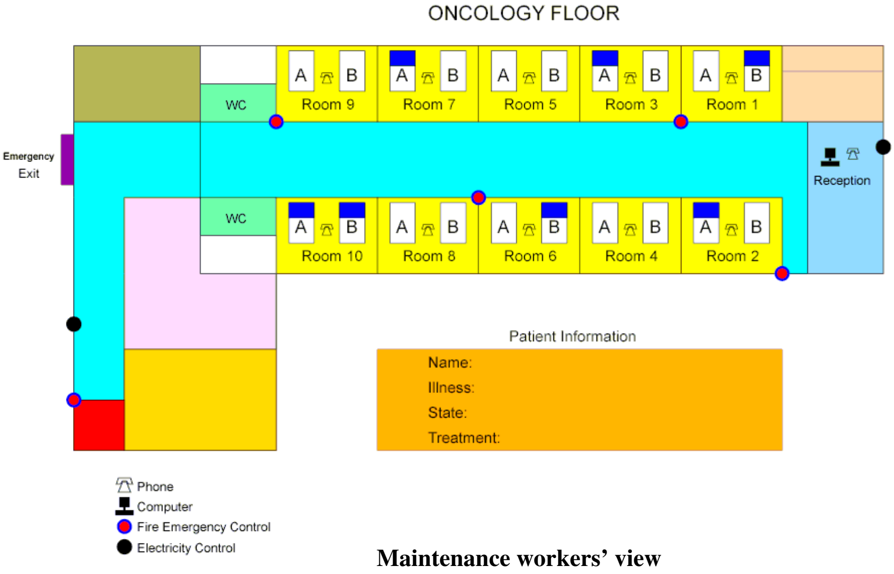
With this expressive way of referring to subjects and objects, it worth noticing how the combined use of positive and negative authorizations result convenient for the specification of different constraints, providing an additive or subtractive way of defining authorized views. As an example, consider the graphic illustrated in Figure 1 and the user group hierarchy and the users' profiles in Figure 3. Figure 6(a) presents some examples of protection requirements that can be expressed in our model. Also, Figure 6(b) illustrates the view that, according to the given protection requirements, will be returned to the maintenance workers; in the next Section we illustrate the process for obtaining such a view.

4. POLICY ENFORCEMENT

The enforcement algorithm in Figure 7 consists of two main phases: *node labeling* (steps 1-4) and *tree transformation* (steps 5-8). Node labeling takes as input a user request and the DOM tree of the SVG document. Then, it evaluates the authorization rules to be enforced and selectively assigns a plus or minus sign to the nodes of the DOM tree. Subsequently, the tree transformation phase takes the labeled DOM tree as input and transforms it into another valid DOM tree. The result is

```
[Rule 1] Everybody can see the emergency exits
   <subject><id value=‘Users’/></subject>
   <object><refer =‘type.emergencyexit’/></object>
   <sign value=‘+’/>

[Rule 2] Everybody can see the content of any room in
          the public area
   <subject><id value=‘Users’/></subject>
   <object><refer =‘id.PublicArea’/></object>
   <sign value=‘+’/>

[Rule 3] Everybody can see the perimeter of any room in
          the private area
   <subject><id value=‘Users’/></subject>
   <object>
      <refer =‘g[@id=‘oncologyfloor’]//g’/>
      <cond>inside(id.PrivateArea)</cond>
   </object>
   <sign value=‘+’/>

[Rule 4] Medical staff can see the content of any room
          in the private area
   <subject><id value=‘MedicalStaff’/></subject>
   <object><refer =‘id.PrivateArea’/></object>
   <sign value=‘+’/>

[Rule 5] Only members of the NonMedicalStaff whose job is
          ‘maintenance worker’ can see the fire emergency
          and electricity controls
   <subject>
      <id value=‘NonMedicalStaff’/>
      <cond>job[@value=‘maintenance worker’]</cond>
   </subject>
   <object><refer =‘type.electricitycontrol’/></object>
   <sign value=‘+’/>

   <subject>
      <id value=‘NonMedicalStaff’/>
      <cond>job[@value=‘maintenance worker’]</cond>
   </subject>
   <object><refer =‘type.fire emergency’/></object>
   <sign value=‘+’/>

[Rule 6] Doctors with specialty ‘oncology’ can read patient
          information; everybody else is explicitly
          forbidden
   <subject>
      <id value=‘Doctors’/>
      <cond>specialty[@value=‘oncology’]</cond>
   </subject>
   <object><refer =‘type.patientinformation’/></object>
   <sign value=‘+’/>

   <subject><id value=‘Users’/></subject>
   <object><refer =‘type.patientinformation’/></object>
   <sign value=‘−’/>
```

(a)



Maintenance workers’ view

(b)

Figure 6. An example of authorization rules (a) and of view (b) on the SVG document in Figure 1

a view of the SVG document containing only the elements that the requestor is entitled to access.[2]

[2] For simplicity, here we assume that conflicts of authorization referred to a same element and with disjoint subjects are solved by applying the most specific takes precedence principle (i.e., the authorization with the subject more specific with respect to the user group hierarchy

Algorithm 1 *Enforcement algorithm*

```
1 Determine the set Applicable_authorizations of authorizations applicable to the requestor. These are all the authorizations for which the requestor is a member (possibly proper) of the subject identity (id) and for which requestor's profile satisfies the subject expression (subj-expr).
2 Evaluate the object expressions in every authorization in Applicable_authorizations (e.g., resolving them into suitable XPath queries), and label the corresponding SVG elements with the authorization subject identity (id) and the sign of the authorization.
3 If an element has more than one label eliminate all labels whose subject identity is a super-group of the subject identity in another label (most specific takes precedence).
4 If an element remains with more than one label and they are of the same sign, assume that sign for the element. If labels are of different sign assume '−' (denials take precedence).
5 Starting from the root, propagate each label on the DOM tree as follows:
    (a) one step upward to the father node, provided the father node is a g element while the current node is not.
    (b) downward to descendants. Unlabeled descendants take the label being propagated and propagate it to their children, while labeled ones discard the label being propagated and propagate their own to their children (most specific takes precedence).
6 Discard from the document all subtrees rooted at a node with a negative label.
7 Discard from the document all subtrees whose nodes are all unlabeled nodes.
8 Render the resulting document.
```

Figure 7. Enforcement algorithm

While steps **1-4** solve the problem of determining labels to be given to nodes according to applicable authorizations, step **5** deals with completing the labeling phase by propagating initial labels on the SVG document's DOM tree; such step is specific to the SVG data model and deserves some comments. In a well-behaved SVG document (see Section 3), groups `g` include a `typeElement` attribute. Also, they contain only two children nodes: a definition of their perimeter and a subgroup for the rest of their content. If the perimeter of a well-behaved group node gets a minus label, our upward propagation ensures that the whole group subtree is pruned; if the minus label affects the subgroup node, the empty perimeter is preserved. On the other hand, if the `g` node content is a flat list of SVG elements (a situation which is discouraged but not prevented by current SVG specifications), upward propagation ensures that a minus on each contained node will make the group disappear completely, including the perimeter (alternatively, a log event could be generated instead to alert the application enforcing the policy).

prevails) and the denials take precedence principle for conflicts between authorizations whose subjects are incomparable.

In this way *consistency* of the view is preserved (i.e., our algorithm will not produce non-sense views where, for example, a door shows up in an empty area). Of course, intelligent coordinates-based checking could solve the problem by distinguishing objects lying "inside" and "outside" other objects, but this would require our enforcement engine to provide complex query computing capabilities. While such capabilities may well turn out to be important in the long run, we chose not to include them in the present version of our system. Note also that non-group nodes (satisfying the FPL language predicates, if any) will be simply pruned together with their subtrees; no guarantee is offered in this case that the image semantics will be preserved.

5. CONCLUDING REMARKS

We have presented a technique for fine-grained feature protection of XML-based graphics formats. While we developed this technique mainly for controlled dissemination of graphical information representing confidential or sensitive data, other interesting applications of feature protection techniques, mainly related to digital rights management and intellectual property protection, are currently under discussion. These applications promise to be an interesting direction in which extends the proposal presented in this paper.

References

[1] P. Bonatti, E. Damiani, S. De Capitani di Vimercati, and P. Samarati. An access control system for data archives. In *16th International Conference on Information Security*, Paris, France, June 2001.

[2] P. Bonatti, E. Damiani, S. De Capitani di Vimercati, and P. Samarati. A component-based architecture for secure data publication. In *17th Annual Computer Security Applications Conference*, New Orleans, Louisiana, December 2001.

[3] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. A fine-grained access control system for XML documents. *ACM Transactions on Information and System Security (TISSEC)*, 5(2):169–202, May 2002.

[4] S. Jajodia, P. Samarati, M.L. Sapino, and V.S. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems*, 26(2):18–28, June 2001.

[5] 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.

[6] World Wide Web Consortium. *Scalable Vector Graphics (SVG) 1.0 Specification*, September 2001.

[7] World Wide Web Consortium (W3C). *Document Object Model (DOM)*. http://www.w3.org/DOM/.

[8] World Wide Web Consortium (W3C). *XSL Transformations (XSLT).* http://www.w3.org/Style/XSL.

[9] World Wide Web Consortium (W3C). *XML Path Language (XPath) 2.0*, December 2001. http://www.w3.org/TR/xpath20.

[10] J. Ze Wang, M. Bilello, and G. Wiederhold. Textual information detection and elimination system for secure medical image distribution. In *Proc. of the 1997 American Medical Informatics Association (AMIA'97) Annual Fall Symposium (formerly SCAMC)*, Nashville, Tennessee, October 1997.

[11] J. Ze Wang and G. Wiederhold. System for efficient and secure distribution of medical images on the internet. In *Proc. of the 1998 American Medical Informatics Association (AMIA'98) Annual Fall Symposium*, Orlando, Florida, November 1998.