Developing Filtering Techniques for Securing Vector Graphics Images Applied to Ubiquitous Patient Records

Dr. Sabah MOHAMMED, Dr. Jinan FIAIDHI and Ahmed Sabbir ARIF

Department of Computer Science, Lakehead University,
955 Oliver Road, Thunder Bay, Ontario P7B 5E1,
Canada
http://flash.lakeheadu.ca/~mohammed
http://flash.lakeheadu.ca/~jfiaidhi

Abstract- In this article we are describing the security challenges with the use of SVG Vector Graphics for ubiquitous patient records. Moreover, we are presenting an architecture that incorporates security mediators in the form of SVG filers to provide a highly flexible approach for accessing Electronic Patient Records. The SVG filters are based on the SAX primitives to pushes pieces of the SVG to the encryption/decryption handlers. The SAX handlers can filter, skip tags, or encrypt tags partially or universally at any time from the stream of the SVG. The SVG encryption/decryption techniques used by the SAX filters implements the standard specification of the W3C XML Encryption.

Key-Words- Vector Graphics Images, SVG, Ubiquitous Patient Records, SAX Filters, XML Encryption.

1. INTRODUCTION

Healthcare today is moving away from the tethered domain and becoming diffused into an environment rich with ubiquitous and portable digital devices. In this evolving environment, the need to deliver information such as Electronic Patient Records (EPR) at the point – of – care is a prime factor in managing the healthcare system efficiently. This however presents serious security challenges in ubiquitous environments where protecting the confidentiality of the information, while at the same time allowing authorized user to access it conveniently is the core issue in the paradigm. However, ubiquitous computing based on XML has the potential to improve many workflows in Health Care. In particular, XML promises a means of providing flexible, incremental employment of electronic systems, and also provides a means for easy communication between disparate binary electronic systems. Ultimately and ideally the whole healthcare record system will be based on a non-proprietary mark-up language (XML), but till that day comes we need to develop effective XML security that can work within variety of environments. In this direction, several organizations have produced standards for XML security and XML patient records. The most visible of these are the Health Level 7 (http://www.hl7.org/), the W3C XML Encryption and Signature (www.w3c.org), EHRcom (http://www.centc251.org/), OpenEHR (http://www.openehr.org/), MedBiquitous (http://www.medbiq.org/), HandHeldMed (http://www.patienttracker.com/products.htm*)*, DICOMX (http://142.104.48.20/Benjamin.Jung/), *Wireless MediCenter* (http://www.wirelessmedicenter.com/mc/glance.cfm) the *m-care* (http://www.m-care.co.uk/tech.html*) PatientKeeper (*http://www.patientkeeper.com/products.html) and *PocketMD* (http://www.pocketmd.com). Furthermore, motivations such as patient privacy protection and recent laws like the US Health Insurance Portability and Accountability Act (HIPAA), the US President Executive Order (13335 of April 2004) on the migration to EPR, and the Canada Health Act make security a fundamental concern within the healthcare industry.

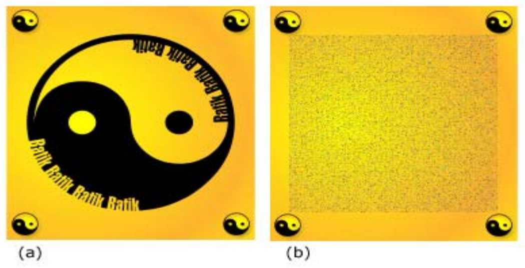
Indeed, the use of XML is not limited for the representation of textual documents, but it can also be used to represent each document in the processing of medical tests and imaging. Once an application is built on XML such as a patient record standard, then a wide range of other XML standards can be brought to bear such as using XML parsing based on DOM or SAX models, or even as to represent and process graphical images/multimedia using the W3C SVG vector graphics standard. Our objective in this article is to achieve secure sharing and accessing of image/multimedia medical information on the Internet or on ubiquitous devices while avoiding violations of security or privacy. In this direction we are investigating, developing, and testing filtering capabilities to complement other means of controlling release of SVG medical documents to individuals and organizations outside of the direct health-care delivery setting. Our specific focus is on the use of effective SVG filtering of the information contained in images that are part of an electronic medical record.

However, since SVG graphics is in the end just text (excluding the cases of embedded raster graphics), the source is "open." This raises potential security-related concerns in form of copyright infringement and misuse. Although these issues already exist with other text-based formats on the Web (e.g. HTML, CSS, XML and JavaScript), there is a dire need for a different solution; that take into account the nature of the vector graphics. A possible direct solution can be based on the use a compression mechanism that converts vector graphics into binary with an "encryption" to increase its security. The problem with this approach is that it is contrary to the advantage of including metadata information in SVG, if the markup is "scrambled" in any way. Even if the metadata information can be extracted, the authoring and rendering (not to mention searching) software are imposed with the additional burden of supporting that compression mechanism. Even the use of a compression scheme like XMLZip or XMill which leaves the metadata information intact has very little success [1]. The other fine solution is to keep the SVG structure intact and try to filter in and out information that need to be secured only. In this direction there are two main approaches. One approach which uses an access control map to allow only authorized accesses to particular parts of the vector graphics [2]. Unfortunately, controlling access requires a perfect organization of the internal data in an enterprise. In many practical cases this requirement cannot be fulfilled, since it implies a radical restructuring of all internal information services. The cost of aligning all internal data to deal with external access privileges is not only costly from the systems point-of-view, but also for all internal users of information systems, who now must file all data according to external requirements that are normally none of their concern. The second approach is more comprehensive and relies on filtering the graphical features of the vector graphic which are in the form of contours (i.e. SVG paths) and try to approximate it for compression and security purposes. In this direction, researchers either use direct approximations based on and cubic bezier, and elliptical curves [3] for compression purposes only or use triangulation techniques based on filtering in the frequency domain using DDT (Data Dependent Triangulation) or WBT (Wavelet Based Triangulation) for compression and security purposes [4]. . Results based on the direct approximation are not very good: files size are too high, images are blurred, with a bad "blocked" effect and colors are too much different from the original image [5]. Triangulation on the other hand requires a highly expensive iterative algorithm that segments the vector image according to complicated wavelet transformation which is applied to a Fourier-transformed representation of the image [6].

In this article we are concerned with SVG *image filters* which transform securely an SVG image in a predictable fashion by recognizing the SVG structure. For this purpose, our image filters replies on utilizing a SAX parsing algorithm along with an implementation to the W3C XML Encryption policy.

2. The W3C SVG Visual Filters:

Visual filters are a familiar concept if you've ever used graphics programs: You take a raw image and tell your program to put a little blur on it, or you might reverse some of the image itself. In this direction W3C [7] recommended certain SVG filters which consist of a series of graphics operations that are applied to a given source graphic to produce a modified graphical result. The result of the filter effect is rendered to the target device instead of the original source graphic. Filter effects are defined by 'filter' elements. To apply a filter effect to a graphics element or a container element, you set the value of the 'filter' property on the given element such that it references the filter effect. Each 'filter' element contains a set of filter primitives as its children. Each filter primitive performs a single fundamental graphical operation (e.g. blur) on one or more inputs, producing a graphical result. Because most of the filter primitives represent some form of image processing, in most cases the output from a filter primitive is a single RGBA image. When applied to container elements such as 'g', the 'filter' property applies to the contents of the group as a whole. Typically, the graphics commands are executed as part of the processing of the referenced 'filter' element via use of the keywords SourceGraphic or SourceAlpha. Filter effects can be applied to container elements with no content (e.g., an empty 'g' element), in which case the SourceGraphic or SourceAlpha consist of a transparent black rectangle that is the size of the filter effects region.SVG Filter primitives such as 'feGaussianBlur', 'feOffset', 'feSpecularLighting', 'feComposite', 'feMerge', ' feTurbulence ', 'feConvolveMatrix', 'feDiffuseLighting', 'feDisplacementMap', 'feFlood', 'feImage', 'feMorphology', and 'feTile' takes input SourceAlpha, which is the alpha channel of the source graphic. The result is stored in a temporary buffer which then can be used by another SVG filter primitive to add more graphical effects. Figure 1 illustrates the use of an SVG filter (feTurbulence) to filter out part of the original image as described by the rectangular SVG tag.



```
<svg width="450" height="325" viewBox="0 0 450 325">
  <title>Example feTurbulence </title>
    <g style="font-family:Verdana; text-anchor:middle">
      <defs>
<filter id="Del" filterUnits="objectBoundingBox" x="0%"
y="0%" width="100%"   height="100%">
<feTurbulence type="turbulence" baseFrequency="0.05"
numOctaves="2"/>
</filter>
      </defs>
……
          </g>
</svg>
```

(c)

Figure 1: (a) Original SVG Image, (b) SVG image after applying feTurbulence filter, (c) SVG code.

Although, the W3C SVG filters can be used to hide or change some of the SVG document contents using a wide variety of server side SVG generators that can add SVG filters on the fly (e.g. AgileBox, DataSlinger, CGI Perl Scripts), the usage of W3C SVG filters can not secure SVG images as the receiver can delete these filters from the SVG source and view the original images.

3. XML Encryption Standard for SVG Security:

With the increasing importance and widespread distribution of SVGs the protection of the intellectual property rights of the owner for their media has become increasingly significant. SVG can be copied and widely distributed without any significant loss of quality. Protecting the property rights of the owners' data is therefore an increasingly important capability. In this direction, the W3C has provided the specifications for the security of any XML compliant data (e.g. SVG) based on what is called "XML Encryption"(http://www.w3.org/Encryption/2001/).
Although, there are many applications for such specifications given the increasing importance of XML on the Internet and Web, these implementations are based on heavyweight DOM APIs (e.g. XSS4J, Apache XML Security API, XMLSec, XMLDSig), which can not yield acceptable performance for ubiquitous applications (e.g. mobile, SmartCard)[8].

Generally speaking, there are two programming models for working with XML infosets: document *streaming (SAX)* and the *document object model* (DOM). The *DOM* model involves creating in-memory objects representing an entire document tree and the complete infoset state for an XML document. Once in memory, DOM trees can be navigated freely and parsed arbitrarily, and as such provide maximum flexibility for developers. However the cost of this flexibility is a potentially large memory footprint and significant processor requirements, as the entire representation of the document must be held in memory as objects for the duration of the document processing. This can cause major performance degradation when used for ubiquitous devices. *However, SAX streaming* refers to a programming model in which XML infosets are transmitted and parsed serially at application runtime, often in real time, and often from dynamic sources whose contents are not precisely known beforehand. Moreover, stream-based parsers can start generating output immediately, and infoset elements can be discarded and garbage collected immediately after they are used. Thus, streaming provides a smaller memory footprint, reduced processor requirements, and higher performance in processing multimedia. With SAX streaming we choose to access the XML multimedia, not as a tree of nodes, but as a sequence of events! This makes SAX faster for highly constrained and ubiquitous devices [9] but it necessitates the following things:

- creation of your own custom object model
- creation of a class that listens to SAX events and properly creates your object model.

At its core, SAX is based on just two interfaces, the `XMLReader` interface that represents the parser and the `ContentHandler` interface that receives data from the parser. As SAX is the most reliable streaming API for parsing XML, it becomes the most visible alternative for implementing SVG security within mobile/ubiquitous environments.

4. The Proposed SVG Security Architecture:

Our architectural model is based on the concept of a security mediator, called SAX Filters, that enables legitimate external customers to gain remote electronic access to medical information residing in a medical institution, while inhibiting the release of content that cannot be released, even when the accessors appear to be authorized. Such a security mediator is typically placed into the firewall surrounding the institution's internal database activities. A SAX filter is simply a class that is passed as the event handler to another class that generates SAX events, then forwards all or some of those events on the next handler (or filter) in the processing chain. A filter may prune the document tree by not forwarding events for elements with a given name (or that meet some other condition), while in other cases, a filter might generate its own new events to add parent or child elements to certain elements the existing document stream. Also, element attributes can be added or removed or the character data altered in some way. SAX filters often perform only a single, simple task, but when piped together they are capable of complex tasks. The basic structure of a SAX filter is based on an `XMLReader` that receives already parsed events from another `XMLReader`. Figure 2 diagrams the course of SVG processing with a SAX filter. A client application instructs the filter, represented in SAX by an `XMLFilter` object, to read the text of an XML document. The filter then instructs the parser to read the text of an XML document. As it reads, the parser calls back to the filter's `ContentHandler`. The filter's `ContentHandler` then calls back to the client application's `ContentHandler`.

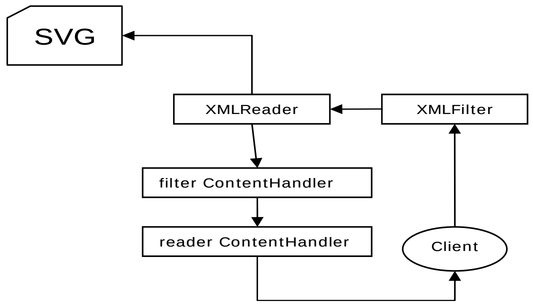


Figure 2: The course of SVG processing with a SAX filter.

Since the filter sits in the middle between the real parser and the client application, it can change the stream of events that gets passed back and forth between the two. For example, it can add new shapes to SVG on the fly. It can add namespaces to elements and attributes that don't normally have them. It can work with the stream without actually changing the data itself. And as we will see in later sections, it can call a cryptography library to encrypt elements, decrypt encrypted elements, sign a document or verify the signature.

Using this idea of a SAX filter, we developed a model for protecting SVG multimedia contents. A client application will instructs the SAX Filter to read the text of an SVG multimedia. The filter then instructs the parser to read the text of an SVG document. As it reads, the parser calls back to the filter's ContentHandler. The filter's ContentHandler will contain the processing instructions, which could be an instruction to encrypt/decrypt elements of the SVG the document. The filter's ContentHandler will do the cryptographic tasks and will call back to the client application's ContentHandler.

4.1. Developing the SVG SAX Filter

Our developed SAX filter is called XMLFilter which its interface inherits methods from the XMLReader superinterface. XMLFilter has just two methods, getParent() and setParent(). The parent of a filter is the XMLReader to which the filter delegates most of its work. XMLReader provides fourteen other methods such as getFeature(), setFeature(), getProperty(), setProperty(), setEntityResolver(), getEntityResolver(), setDTDHandler(), getDTDHandler(), setContentHandler(), getContentHandler(), setErrorHandler(), getErrorHandler() and parse().

The developed SVG Security prototype involves two SAX filters, for encrypting and decrypting (FilterEncryption and FilterDecryption). The main class of the SVG security prototype is called SVGEncryption which calls instances of these SVG filters and an instance of a real parser, then passed the real parser to the filter's `setParent()` method:

```
String encryptFilter = "FilterEncryption";
String decryptFilter = "FilterDecryption";
// Encryption filter
XMLFilter filter =
(XMLFilter)Class.forName(encryptFilter).newInstance(
);
filter.setParent(XMLReaderFactory.createXMLReader(
));
// Decryption Filter
XMLFilter filter = (XMLFilter)
Class.forName(decryptFilter).newInstance();
filter.setParent(XMLReaderFactory.createXMLReader(
));
...
```

By doing this it is confirmed that the client application only interacts with these filters. It forgets that the original parser exists. Going behind the back of the filter, for instance, by calling `setContentHandler()` on `parser` instead of on `filter`, runs the risk of confusing the filter by violating constraints it expects to be true. All such occasions was carefully avoided. Actually, the XMLFilter interface filters are called from the client application to the parser. Most events are passed in the other direction from the parser to the client application through the various callback interfaces, especially ContentHandler. Hence, the XMLFilter is set up to filter calls from the client application to the parser, but not calls from the parser to the client application. To achieve this type of communication, the setContentHandler() method is replaced with two method calls (HandlerDecryption, HandlerEncryption) so that the handlers receive the callback events from the parent parser. These methods either passes them along or passes something different as instructed to the client application's handler methods. The following illustrates how these two methods are called via the setContentHandler() method:

```
...
public void setContentHandler(ContentHandler
handler)
```

```
{ parent.setContentHandler(new
HandlerEncryption(handler));}
...
public void setContentHandler(ContentHandler
handler)
{ parent.setContentHandler(new
HandlerDecryption(handler));}
...
```

4.1.1 Handler for FilterEncryption: An Example of one of the SAX Filters

The first thing the HandlerEncryption does, it asks the users to provide with some information (e.g Encrption Key). When the information collection is done, it goes straight to the business. To do the encryption the filter content handler does the followings:

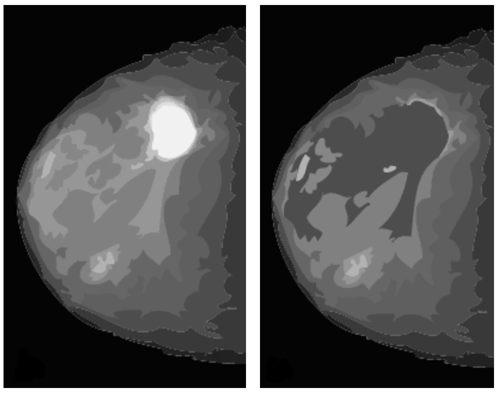
1. When the parser encounters the start-tag and end-tag that needs to be encrypted it changes the tag name to <EncryptedData> and </EncryptedData>
2. When the parser encounters the element that needs to be encrypted it sends the element content to Encrypter class to do the encryption and receives the cipher text.
3. Creates new attributes of EncryptedData (to save the cipher and other encryption information).

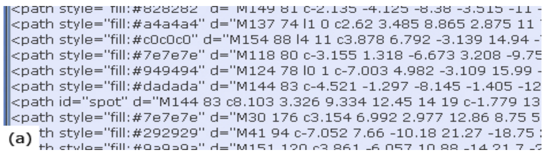
The handler makes sure that the other attribute doesn't get lost or corrupted. This handler can encrypt all the SVG tags (e.g. all path tags), any particular SVG tag (e.g. only a particular path tag), or even a portion of a particular path (e.g. "horizontal lineto", "smooth curveto", etc).

5. EXPERMENTATIONS

Figure 3 shows an SVG mammogram image (part a) with a white spot indicating an abnormality, where (part b) of the image shows how our SAX encryption worked on the path surrounding that spot and encrypt it to the dark grey area. The partial area that needs to be encrypted can be identified by using an identifier like id="spot" and the SAX encryptor will replace it with an encrypted text identified by EncryptedData.

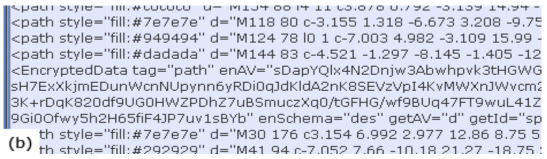
Moreover, the SAX filter has been used effectively to encrypt and decrypt SVG patient records using a Web based applet (Figure 4). However, there are many implementation issues remain to be addressed at our next research especially when SVG patient record is processed under P2P (e.g. JXTA), J2ME or JavaCard environments [10]. The authors are also conducting additional experimentations to use the new javacard.security [11] API instead the current java.security API [12] as well as the StAX [13] primitives instead of the SAX.





(a)

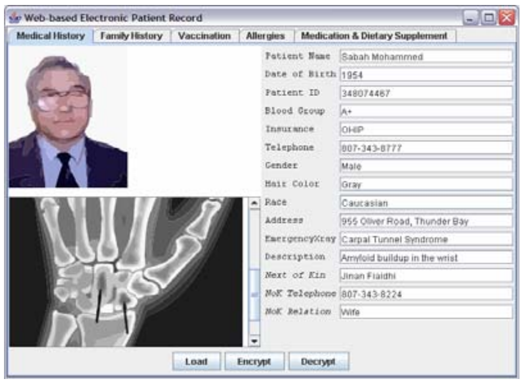
```
<path style="fill:#828282" d="M149 81 c-2.135 -4.125 -8.38 -3.515 -11 -
<path style="fill:#a4a4a4" d="M137 74 l1 0 c2.62 3.485 8.865 2.875 11
<path style="fill:#c0c0c0" d="M154 88 l4 11 c3.878 6.792 -3.139 14.94 -
<path style="fill:#7e7e7e" d="M118 80 c-3.155 1.318 -6.673 3.208 -9.75
<path style="fill:#949494" d="M124 78 l0 1 c-7.003 4.982 -3.109 15.99 -
<path style="fill:#dadada" d="M144 83 c-4.521 -1.297 -8.145 -1.405 -12
<path id="spot" d="M144 83 c8.103 3.326 9.334 12.45 14 19 c-1.779 13
<path style="fill:#7e7e7e" d="M30 176 c3.154 6.992 2.977 12.86 8.75 5
  th style="fill:#292929" d="M41 94 c-7.052 7.66 -10.18 21.27 -18.75
  th style="fill:#9a9a9a" d="M151 120 c3.861 -6.057 10.88 -14.21 7 -
```



(b)

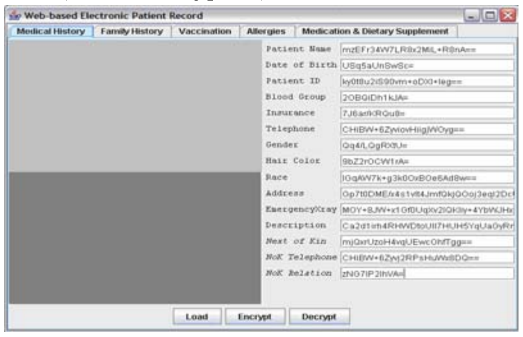
```
<path style="fill:#c0c0c0" d="M154 88 l4 11 c3.878 6.792 -3.139 14.94 -
<path style="fill:#7e7e7e" d="M118 80 c-3.155 1.318 -6.673 3.208 -9.75
<path style="fill:#949494" d="M124 78 l0 1 c-7.003 4.982 -3.109 15.99 -
<path style="fill:#dadada" d="M144 83 c-4.521 -1.297 -8.145 -1.405 -12
<EncryptedData tag="path" enAV="sDapYQlx4N2Dnjw3Abwhpvk3tHGWG
sH7ExXkjmEDunWcnNUpynn6yRDi0qJdKldA2nK8SEVzVpI4KvMWXnJWvcm2
3K+rDqK820df9UG0HWZPDhZ7uBSmuczXq0/tGFHG/wf9BUq47FT9wuL41Z
9Gi0Ofwy5h2H65fiF4JP7uv1sBYb" enSchema="des" getAV="d" getId="sp
  th style="fill:#7e7e7e" d="M30 176 c3.154 6.992 2.977 12.86 8.75 5
  th style="fill:#292929" d="M41 94 c-7.052 7.66 -10.18 21.27 -18.75
```

Figure 3: Encrypting a Partial SVG Image.



Patient Name	Sabah Mohammed
Date of Birth	1954
Patient ID	348074467
Blood Group	A+
Insurance	OHIP
Telephone	807-343-8777
Gender	Male
Hair Color	Gray
Race	Caucasian
Address	955 Oliver Road, Thunder Bay
EmergencyXray	Carpal Tunnel Syndrome
Description	Amyloid buildup in the wrist
Next of Kin	Jinan Fiaidhi
NoK Telephone	807-343-8224
NoK Relation	Wife

(Before Encryption)



Patient Name	mzEFr34W7LR8x2ML+R8nA==
Date of Birth	USq5aUnSwSc=
Patient ID	ky0t8u2iS90vm+oDXI+leg==
Blood Group	2OBGiDh1kJA=
Insurance	7J6arlKRQu8=
Telephone	CHIBW+6ZwiovHiigWOyg==
Gender	Qq4/LOgRX0U=
Hair Color	9bZ2rOCW1rA=
Race	lGqAW7k+g3k0OxBOe6Ad8w==
Address	Op7t0DME/x4s1v84JmfQkjQOoj3eqt2Dc
EmergencyXray	MOY+8JW+x1Gf0UqXv2lQH3iy+4YbWJHx
Description	Ca2d1xh4RHWDtoUIl7HUH5YqUa0yRr
Next of Kin	mjQxrUzoH4vqUEwcOhfTgg==
NoK Telephone	CHIBW+6Zwj2RPsHuWx8DQ==
NoK Relation	zNG7IP2lhVA=

(After Encryption)

Figure 4: A Screen Shot of the Web Based Prototype for Encrypting/Decrypting SVG Patient Medical Records.

6. CONCLUSIONS

Current XML security implementations are based on the DOM model. With ubiquitous devices we need more effective model especially for XML multimedia security. SAX represents a solution along with the new SVG multimedia format. This article introduced the steps for developing SAX filters for implementing the XML Encryption on SVG. The SAX Encryption/Decryption enables us to encipher any SVG tag (s) or even to encipher an attribute within a specific tag (e.g. to encipher line (L) within the path tag).

REFERENCES

[1] M. Cokus and D. Winkowski, "XML Sizing and Compression Study for Military Wireless Data", XML Conference 2002, Dec. 8-13, 2002, Batimore, MD, USA.

[2] E. Damiani, S. De Capitani di Vimercati, E. Fernández-Medina, 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*, King's College, University of Cambridge, UK, July 29-31, 2002.

[3] P. Kamthan, "**XMLization of Graphics",** Internet Related Technology OnLine Journal, Monday 27th March 2000, http://www.irt.org/articles/js209/.

[4] S. Battiato, G. Barbera, G. Di Blasi, G. Gallo, G. Messina, "*Advanced SVG Triangulation/Polygonalization of Digital Images*", In proceedings of IS&T/SPIE Electronic Imaging 2005.

[5] J. Feng, T, Nishita, X. Jin, Q. Peng, "B-Spline Free-Form Deformation of Polygonal Object as Trimmed Bezier Surfaces", The Visual Computer, Vol.18, No.8, pp.493-510, 2002-12

[6] S. Lee, "*Wavelet-Based Multiresolution Surface Approximation from Height Fields*", Ph.D. Thesis, Virginia Polytechnic Insitute and State University, Blacksburg, February 2002

[7] Matthew Duignan, Robert Biddle, Ewan Tempero, "Evaluating scalable vector graphics for use in software visualisation", Proceedings of the Australian symposium on Information visualisation, Adelaide, Australia, Volume 24, pp. 127 – 136, 2003

[8] Dmitry Pavlov, Jianchang Mao, Byron Dom. "Scaling-Up Support Vector Machines Using Boosting Algorithm," *icpr*, p. 2219, 15th International Conference on Pattern Recognition (ICPR'00) - Volume 2, 2000.

[9] Yanlei Diao, and Michael J. Franklin, "High-Performance XML Filtering: An Overview of YFilter", *IEEE Data Engineering Bulletin* , March, 2003

[10] J. Fiaidhi, S. Mohammed, M. Garg and A. Arif, "Developing a SAX Filtering Intermediary Service for Protecting SVG Multimedia Contents in a Ubiquitous Publish/Subscribe Environment", *Int.Conference on Internet Computing (ICOMP05)*, Las Vegas, USA, June 27-30, 2005.

[11] C. Enrique Ortiz, "An Introduction to Java Card Technology", Sun Developer Network White Paper, September 2003 http://developers.sun.com/techtopics/mobility/javacard/articles/javacard3/

[12] MIT Press, "Java^{TM} Cryptography Architecture API Specification & Reference", *2 May 1997, http://cycleserv2.csail.mit.edu/jdk/guide/security/CryptoSpec.html*

[13] Elliotte Rusty Harold, "An Introduction to StAX", O'Reilly XML.com Newsletter, September 17, 2003 http://www.xml.com/pub/a/2003/09/17/stax.html