Securely Streaming SVG Web-Based Electronic Healthcare Records involving Android Mobile Clients

Sabah Mohammed and Jinan Fiaidhi
Department of Computer Science, Lakehead University,
Thunder Bay, Ontario P7B 5E1, Canada
{mohammed, jfiaidhi}@lakeheadu.ca

Osama Mohammed
Department of Software Engineering, Lakehead University,
Thunder Bay, Ontario P7B 5E1, Canada
omohamme@lakeheadu.ca

***Abstract*— Although Electronic Healthcare Records (EHRs) technology largely facilitates patient care by providing clinicians with the ability to review a more complete medical record, interoperability and privacy issues present significant barriers to their implementation. This article proposes the open source SVG (Scalable Vector Graphics) standard for representing electronic healthcare records for interoperability purpose where security can be enforced using lightweight SAX streaming filters. The SVG filters are based on the Java SAX API to push 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 EHRs. A prototype for implementing the SAX streaming filter is presented along with experiments to test its applicability in a web environment for sharing SVG EHRs on the Android mobile development environment.**

***Index Terms*— *Open Source* EHRs, Semantic Interoperability, SVG, SAX Filters, XML Encryption Standard, Android.**

I. INTRODUCTION

E-health networks can provide more seamless and integrated services to patients and health care workers that are more broadly accessible by leveraging Internet technology and electronic health records. In order to do so, however, issues of security and privacy of personal health information must be addressed [1]. Moreover, healthcare systems globally are challenged by the human and financial resource requirements of an ever growing and aging population. Health promotion and preventative programs along with early and rapid access to treatment are all key factors to improving healthy living. Investments in medical technology to improve the delivery of health care are also a critical consideration and it is here that the mobile Internet has a role to play. Mobile Internet technology has also proven itself invaluable in bringing important medical applications to the point of care [2]. In the past, physicians and healthcare users who required information related to a medication almost always had to wait for the legacy system to provide it in a paper fashion. Healthcare has long relied upon paper based record systems which have become cumbersome and expensive to manage and present significant challenges related to speed of accessibility and security. Thus the emerging benefit of mobile Internet technology to healthcare is to provide mobile access to medical records. Again, using mobile technology means that the treatment process can be sped up and the potential for medical errors can be reduced. With motivations such as patient privacy protection and laws like the US Health Insurance Portability and Accountability Act (HIPAA), the US President Executive Order (13335 of April 2004) on the migration to EHR, the recent President Obama's Healthcare Reform where EHRs is the key for such strategy, the Canada Personal Information Protection and Electronic Documents Act (PIPEDA) and Ontario Personal Health Information Protection Act (PHIPA), make implementations of EHRs and their security a fundamental concern within the healthcare industry.

However, the advantages of mobility and openness offered by the Internet to promote connectivity between healthcare user's devices are not in line with the connectivity between e-health applications. There are many different standards for EHRs (e.g. EN13606, HL7v3 RIM, HL7 CDA) and we need to provide the right harmonization between these different standards to achieve the required compatibility. Although there are many standards development organizations who care about e-health standardization including EHRs (e.g. HL7, CEN, ISO/IEC, ASTM, DICOM, OMG, IHE, IEEE, OASIS, LOINC, SNOMED, WHO, UN/CEFACT, W3C and various universities, research institutes and national standards bodies), much work is still required to resolve several key compatibility issues and gain global acceptance of widely used standards for the

Extended article from an article submitted to the E-health Workshop, Part of MCETCH 2009 Conference, University of Ottawa, Ottawa, Ontario, Canada, May 4-6, 2009.

doi:10.4304/jetwi.1.2.146-152

representation and interchange of shared EHRs. Therefore, the only possible solution to foster more engagement between vendors and the standardization community is to have EHRs as open source and deal with their translation, transcoding and integration through the use of semantic interoperability technologies [3,4,5]. In this paper, we are investigating the issue on how the open source EHRs and the semantic technologies may securely support and promote interoperability among electronic healthcare records systems. Currently, the primitive techniques used for achieving some sort of semantic interoperability are based on XML technologies [6]. Such systems include: Synapses, SynEx, GEHR, GALEN among many others [7,8,9,10,22]. In such systems, the XML based semantic interoperability facilitates the representation, coding, transmission and use of meaning and metadata across health services, between providers, patients, citizens and authorities, research and training [13]. In this direction, any adopted security policy or technique needs to conform to the methods of representation, coding and transmission of XML-based information. For this reason, several e-health organizations developed systems for sharing EHRs where their security is based on the W3C XML Security (www.w3c.org). Among such systems are: EHRcom (www.centc251.org/), OpenEHR (www.openehr.org/), HL7 CDA (*www.hl7.org*), IHE XDS (*www.ihe.net*), HIE RID (www.ihe.net), and DICOMX [23]. However, the type of security adopted is based on identity management techniques such as OpenID (http://openid.net/) and OAuth (http://oauth.net/) which requires trusting a third party. However this becomes more challenging when the objective is collaboration across organizational boundaries. Numerous identity management services as well as access control methods exist for each enterprise and there is a need to develop methods for cross-boundary control. For this reason, some healthcare users do not prefer trusting a third party and prefer to use security that are based on direct trust. Direct trust refers to a situation in which two individuals or organizations have established a trusting relationship between themselves. Whereas third party trust allows individuals to implicitly trust each other without a personal relationship, direct trust is predicated on the existence of a personal or business relationship prior to exchanging secure information. Although, trusting a third party according to some security experts imposes additional security holes and unrequited risks [24], there are many reliable solutions that rely on third parties for identity certification and authentication even within cross-boundary environments [29].

II. The SVG Open Standard

Following the advent of XML in the 1990s, corporate computing customers began to realize the business value in adopting open formats and standardization in the computer products and applications that they relied on. IT professionals benefited from the common data format possible with XML because of its capacity to be read by applications, platforms, and Internet browsers. The use of XML syntax for the exchange of electronic patient records is no exception as it evident in many projects (e.g. Synapses, SynEx [22] and Open XML[25]). However, these efforts have not focused on representing EHRs that are rich with imaging/multimedia data. Certainly the use of XML in these attempts was focused on the representation of the administrative, clinical textual data and the financial transactions related to the patient record. Indeed, the use of XML is not limited for the representation of textual documents, but also it can be used to represent medical tests, imaging and multimedia. In this direction there are varieties of XML compliant formats that can be used to represent imaging/multimedia information besides textual information (e.g. VRML, SVG, MPEG-7). However, selecting any of these formats depends on the quality of information obtained and on how easy it can be retrieved, accessed, filtered and managed. However, VRML and MPEG7 are more dedicated formats for representing multimedia animations only and what is required for representing EHRs must include imaging and textual data. For this purpose, SVG is the only standard format that can be used for such comprehensive representation of patient data including text, imaging and multimedia [13]. Once an application is built using SVG such as the patient healthcare record, a wide range of other XML technologies can be brought to bear its rendering and processing (e.g. CSS, XSLT, XPath, DOM or SAX). The broad support behind SVG comes from its many advantages. SVG has sophisticated graphic features, which is naturally important for a graphic format, but it also benefits from having an XML grammar. SVG has all the advantages of XML, such as internationalization (Unicode support), wide tool support, easy manipulation through standard APIs (e.g. DOM, Batik API) and easy transformation (e.g.XSLT). In the graphical arena and especially compared to raster graphics formats (such as GIF, JPEG or PNG images). SVG has the advantage of being [11,]:

* **Lightweight.** For many types of graphics, an SVG graphic will be more compact than its raster equivalent
* **Interactive**. SVG content can include scripts to enable interaction and animation.
* **Searchable.** Because SVG content is XML, it becomes possible to search the content of an SVG image for text elements, comments or any kind of meta-data.
* **Structured and Accessible**. Graphic objects can be grouped and organized hierarchically
* **Annotatable.** Since SVG is XML, one can annotate any part of the image.

Our objective in this article is to achieve secure sharing and accessing of EHRs represented as XML and SVG for the Web and other ubiquitous environments. In this direction, we are demonstrating the usefulness of the SVG standard in representing medical records that involves medical and radiological imaging such as those involved with several legacy systems (e.g. RIS, PACS and DICOM). Moreover, we are introducing streaming-

based security filtering capabilities to complement other means of controlling the release of SVG EHRs to individuals and organizations on the Web and outside of the direct healthcare delivery settings. However, to use SVG to support EHR, one needs to develop a comprehensive data model to support patient record representation using both textual and graphical primitives on 2-D layout. To achieve this, we need first to adopt an EHRs standard (e.g. HL7 v.3) and there are some tools that may assist healthcare users to create such EHRs according to the selected standard. For example, the Altova MapForce Enterprise Edition 2009 (http://www.altova.com/) toolkit supports mapping to HL7 v.3 standard and convert as well as catalogue the resulting record in a variety of XML-compliant formats including SVG. Having set the SVG EHRs standard and they way it is represented using XML (e.g. RDF), then one can use a powerful query engine to acquire information from the EHRs repositories based on widely used Web servers such as SPARQL (http://www.w3.org/TR/rdf-sparql-query/) and Jena (http://jena.sourceforge.net/). Where SPARQL represents a qualified query and data access specification and Jena provides a Web Server environment that supports SPARQL and includes a rule-based inference engine for further reasoning requirements (www.w3.org/TR/rdf-**sparql**-query/*)* . What concerns us in this article is the security issues related to open source SVG EHRs access on the Web.

III. Securing the Open Source SVG Based EHRs

There are variety of techniques that may be used to secure XML-Based open source data such as the use of XML Security Suite for Java (XSS4J) [12] or the use of the Security Assertion Markup Language (SAML) and its OpenSAML APIs (www.opensaml.org). However, both techniques do not consider the canonical order of the XML-based document. One other possible solution can be based on the use compression techniques that convert vector graphics into binary with an "encryption" to increase its security. However, controlling access to an XML-based document requires a perfect organization of the internal data and the use of a matching metadata. Unfortunately, in both of these cases (i.e. the non-canonical security toolkits and the compression) the order requirement cannot be fulfilled, since their underlying techniques imply a radical restructuring of all internal information structures and may cause scrambling of the metadata. Even if the metadata information can be extracted first (e.g. compression techniques like XMLZip or XMill, which leaves the metadata information intact), such techniques did not succeed [14]. Alternatively, we need to use security techniques that maintain the SVG structure intact and filter only information that need to be secured. Such a security technique can safely digest and sign the canonical form of the SVG document and at the time of verification, the canonical form of the SVG needs to be verified. For this purpose, there are two security techniques, which adhere to the structure of the SVG document: the access control map that allows only authorized accesses to particular parts of the SVG document [15]; and the use of SVG path filters (where such filters apply methods of compression and to alther the SVG path description). The first approach is useful for authentication purposes and cannot be used for general security. The second approach that uses SVG path filters may be built based on utilizing compression and approximation functions (e.g. cubic Bezier [16], triangulation techniques [17]). However, the results of using such functions are not very good: files size are too high, images are blurred, with a bad “blocked” effect and colors are too different from the original image [18, 19]. Alternatively, SVG path filters may be built using certain scripting visual filters or more generally using security filters that are based on streaming the parsing based events. The use of visual filters are a familiar concept if one has ever used browser based scripting languages like JavaScript: The user may take a raw image and tell their browser to put a little blur on it, or they might reverse some of the image itself. In this direction W3C [20] recommended certain SVG visual filters which consist of a series of graphics operations that are applied to a given source SVG document to produce a modified graphical change. The result of the filter effect is rendered to the target device instead of the original source graphic. Filter visual effects are defined by the 'filter' elements within the JavaScript program. 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 RGB image. When applied to container elements such as a graph path 'g', the SVG '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. Although, the W3C SVG visual filters can be used to hide or change some of the SVG document contents using a wide variety of server side SVG generators or scripts that can add SVG filters on the fly (e.g. AgileBox, DataSlinger, CGI Perl Scripts), the usage of W3C SVG visual filters cannot secure SVG documents as the receiver can delete these filters from the SVG

source and view the original images. For this reason we need more generic and secure SVG *filters* which transform securely SVG EHR in a predictable fashion by recognizing and processing the SVG structured elements through parsing and streaming. More specifically we are proposing to use SVG streaming techniques like the Java SAX API along with some standard W3C XML Encryption techniques (www.w3.org/TR/**xml**enc-core/). The use of SAX API helps in parsing and streaming the SVG contents in orderly way as Figure 1 illustrates.

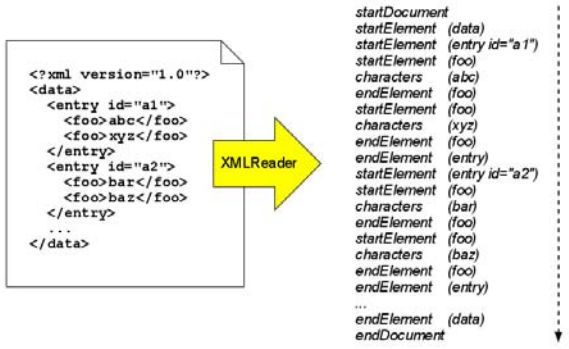

Figure 1: Streaming an XML file using the SAX API.

SAX offers an efficient alternative to processing any XML-based document compared to the Document Object Model (DOM). When you use the DOM to manipulate an XML/SVG file, the DOM reads the file, breaks it up into individual objects; such as elements, attributes, and comments; and then creates a tree structure of the document in memory. The benefit of using the DOM is that you can individually reference and manipulate each object, called a node. However, creating a tree structure for a document, especially a large document, requires a significant amount of memory and utilizes large amount of processing power that cannot be readily available for mobile and highly constrained devices. Unlike the DOM, SAX is stream and event based; it generates events as it finds specific symbols in an XML/SVG document. One advantage of SAX is that it reads a section of an XML/SVG document, generates an event, and then moves on to the next section. Because SAX processes documents in this serial fashion, it uses less memory than the DOM and is therefore better for processing large documents. SAX can create applications that abort processing when a particular piece of information is found. For this reason, the Java SAX API is often associated with mobile applications. This means we can easily use the SAX API with any Java-Enabled mobile device (e.g. Nokia, Windows Mobile, Blackberry or any other Java J2ME mobile device like Android). Android brings attractive design to anyone, regardless of what phone they are using. Additionally, features typically only available on high-end phones, like mapping applications and threaded text messages, are available with Android. Android isn't dependent on device hardware. Instead, it changes which features it offers. iPhone-like features would be reserved for individuals with pricier phones, but those with lesser-priced phones need not worry about lacking applications like mapping and address books. Most importantly with Android, he average consumer does not have to know what open standard (e.g. SVG or XML) is to benefit from it. The use of SAX filters can be extended to be used with the popular Web 2.0 publish and subscribe protocol (e.g RSS) [27]. In short, the Android platform is an open source mobile development platform [28].

IV. Applying the XML Encryption

With the increasing importance and widespread distribution of SVGs, the protection of the information represented by such open-source standard becomes 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 implementations for such specifications given the increasing importance of XML on the Internet and Web, these implementations are based on heavyweight APIs (e.g. Apache XML Security API, XMLSec, XMLDSig), which cannot yield acceptable performance for the Web and ubiquitous applications [20]. Hence, we need to implement a more lightweight implementation to the XML Encryption specification. With SAX streaming, such lightweight implementation can be achieved along with the capability to go through the document structure orderly as a sequence of events. Thus implementing the XML Encryption standard based on SAX requires only listing to these events and handling them. To program systems based on SAX with any Java/J2ME environment like Android, all we need is to implement two interfaces, the `XMLReader` interface that represents the parser and the `ContentHandler` interface that receives data from the parser.

V. The SAX Security Model

Our SAX security programming model is based on the concept of security mediators, called SAX Filters that enable legitimate external customers to gain remote electronic access to the SVG EHR residing in a medical institution, while inhibiting the release of content that cannot be released, even when the requester appears 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 of 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 shows 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/SVG 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: SVG processing with SAX filters.

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 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 EHR contents. A client application instructs the SAX Filter to read the text of an SVG EHR. The filter then instructs the parser to read the text of an SVG EHR. 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 document. The filter's ContentHandler will do the cryptographic tasks and will call back to the client application's ContentHandler.

Developing the SVG SAX Security Filter

Our developed SAX filter is called XMLFilter which inherits its interface 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 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 were 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));}
```

The Security Handler for FilterEncryption

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

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).

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).

Figure 3 illustrates the use of SAX filter in validating authorized and unauthorized access to an SVG EHR for an Android mobile client.

(Authorized Access)

(Unauthorized Access)

Figure 3: Accessing SVG-Based EHRs from a Web URL using the Android Mobile Client.

VI. Conclusions

This article proposes the open source SVG standard for representing EHRs for interoperability purpose where security can be enforced using a lightweight SAX filters. The SAX filters implements the XML Encryption specification standard as introduced by the W3C. The steps for developing these SAX filters are introduced. The SAX Filter Encryption/Decryption services enable users to encipher SVG EHRs as whole or partially for any SVG tag(s) or attribute(s) within a specific tag. A prototype has been developed based on Android mobile environment to test the applicability of using the SAX filters for securely sharing SVG EHRs among trusted parties on the Web and via mobile clients. The SAX filters enable each trusted party to authenticate and access EHRs published by the other trusted party.

Acknowledgment

This research is funded by the first two authors NSERC Discovery Grants. The authors would like to thank Mr. A. Arif for implementing the Java SAX Filter of the desktop client [11,26].

References

[1] L. Peyton, J. Hu, C. Doshi, P. Seguin**, “**Addressing Privacy in a Federated Identity Management Network for EHealth”**,** IEEE Eighth World Congress on the Management of eBusiness (WCMeB 2007)

[2] P. De Toledo et. al., “Interoperability of a Mobile Health Care Solution with Electronic Healthcare Record Systems”, IEEE 28th Annual International Conference of Engineering in Medicine and Biology Society, 2006. EMBS APOS 06. Aug. 30 2006-Sept. 3 2006, pp5214 – 5217.

[3] J.T. Fernández-Breis et al, “Using semantic technologies to promote interoperability between electronic healthcare records’ information models”, Proceedings of the 28th IEEE EMBS Annual International Conference, New York City, USA, Aug 30-Sept 3, 2006, pp2614-2617

[4] D. Lopez and B. Blobel, “Semantic interoperability between clinical and public health information systems for improving public health services”. PubMed Journal, 2007;127: pp256-267. Available Online: .http://www.ncbi.nlm.nih.gov/pubmed/17901617

[5] A. Aguilar, “Semantic Interoperability in the context of e-Health”, Research Seminar, DERI Galway, December 15th, 2005. www.m3pe.org/seminar/aguilar.pdf

[6] D. G Katehakis et al, “An Infrastructure for Integrated Electronic Health Record Services: The Role of XML (Extensible Markup Language)”, Journal of Medical Internet Research (http://www.jmir.org), 17.3.2001.

[7] Health Level 7, SGML/XML Special Working Group. Patient Record Architecture (PRA), (2001 Jan 22). http://www.hl7.org/special/committees/sgml/sgml.htm

[8] Comité Européen de Normalisation (CEN), Technical Committee 251, Working Group I. Information Models. 2001, http://www.centc251.org/

[9] American Society for Testing and Materials (ASTM), subcommittee E31.25. 2001, http://www.astm.org/COMMIT/COMMITTEE/E31.htm

[10] M. Sanromà, A. Mateu and K. Oliveras, “Survey of Electronic Health Record Standards”, Research Group on

Artificial Intelligence (BANZAI), Document from the Research Project K4CARE (IST-2004-026968 K4CARE).
[11] J. Fiaidhi, S. Mohammed, M. Garg and A. Arif (2005), "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.
[12] B. Siddiqui, Secure XML messaging with JMS, Part 2: Using XSS4J to implement XML Security, IBM Research Journal, 21 Feb 2000, http://www.ibm.com/developerworks/edu/x-dw-x-secmes2-i.html.
[13] Semantic-HEALTH-Report, Semantic Interoperability for Better Health and Safer Healthcare: Deployment and Research Roadmap for Europe, Ref.: Plan-Publi 2009.2098, Catalogue number : KK-80-09-453-EN-C, ISBN-13 : 978-92-79-11139-6, Jan. 2 0 0 9.
[14] M. Cokus and D. Winkowski (2002), "XML Sizing and Compression Study for Military Wireless Data", XML Conference 2002, Dec. 8-13, 2002, Batimore, MD, USA.
[15] E. Damiani, S. De Capitani di Vimercati, E. Fernández-Medina, P. Samarati (2002), "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.
[16] P. Kamthan (2000), "XMLization of Graphics**",** Internet Related Technology OnLine Journal, Monday 27th March 2000, http://www.irt.org/articles/js209/
[17] S. Battiato, G. Barbera, G. Di Blasi, G. Gallo, G. Messina (2005), "Advanced SVG Triangulation/Polygonalization of Digital Images", In proceedings of IS&T/SPIE Electronic Imaging 2005.
[18] J. Feng, T, Nishita, X. Jin, Q. Peng (2002), "B-Spline Free-Form Deformation of Polygonal Object as Trimmed Bezier Surfaces", The Visual Computer, Vol.18, No.8, pp.493-510, 2002-12.
[19] S. Lee (2002), "*Wavelet-Based Multiresolution Surface Approximation from Height Fields*", Ph.D. Thesis, Virginia Polytechnic Insitute and State University, Blacksburg, February 2002
[20] M. Duignan, R. Biddle and E. Tempero, "Evaluating scalable vector graphics for use in software visualization", Proceedings of the Australian symposium on Information visualization, Adelaide, Australia, Volume 24, pp. 127 – 136, 2003
[21] F. Ipfelkofer, B. Lorenz and H. Ohlbach, "Ontology Driven Visualisation of Maps with SVG –An Example for Semantic Programming", IEEE Proceedings of the Information Visualization (IV'06), 2006
[22] B. Jung and J. Grimson , "Synapses/SynEx goes XML", In: Peter K, Blaz Z, Janez S, Marjan P, Rolf E, editor. In Proceedings of the Medical Informatics Europe '99 Conference (MIE99): August, 1999; Slovenia, Ljubljana. IOS Press,pp. 906–911.
[23] B. Jung, "DICOM-X - seamless integration of medical images into the HER", 18th IEEE Symposium onComputer-Based Medical Systems,June 23-24 June 2005: 203- 207
[24] N. Szabo, "Trusted Third Parties Are Security Holes", White Paper,2005, Available Online: http://szabo.best.vwh.net/ttps.html
[25] T. Pattison et. al, "Using Office Open XML Formats to Support Electronic Health Records Portability and Health Industry Standards" ,October 2007,MSDN Office Developer Center. Available Online: http://msdn.microsoft.com/enus/library/bb879915.aspx
[26] S. Mohammed, J. Fiaidhi and A. Arif, "Developing filtering techniques for securing vector graphics images applied to ubiquitous patient records," Proceedings of the 5th WSEAS International Conference on Telecommunications and Informatics, Istanbul, Turkey, May 27-29, 2006 (pp139-144)
[27] Tane Piper, "Writing a SAX-based RSS and Podcast Parser", Learning Android, 18/02/2009. http://learningandroid.org/tutorial/2009/02/writing-sax-based-rss-and-podcast-parser,
[28] Michael Galpin, "Working with XML on Android: Build Java applications for mobile devices". IBM Research Journal, 23 Jun 2009, http://www.ibm.com/developerworks/library/x-android/
[29] Oluwafemi Ajayi, Richard Sinnott, Anthony Stell,Dynamic Trust Negotiation For Flexible e-Health Collaborations, Mardi Gras Conference '2008, Jan 31 – Feb 2, Baton Rouge, Louisiana, USA.

Sabah Mohammed is a Full Professor with the Department of Computer Science, Lakehead University, Ontario, Canada. Dr. Mohammed is also an Adjunct research Professor with the University of Western Ontario, Canada. His research interests includes: Medical informatics, Web Oriented Architectures, Open Source Multimedia Protection and Web Intelligence. Dr. Mohammed holds two graduate degrees in Computer Science from Glasgow University (MSc-1981) and from Brunel University (PhD-1986). Dr. Mohammed is a Professional member of ACM, member of the British Computer Society (MBCS), member of the Canadian Information Processing Society (ISP) and a Professional Engineer (PEng).

Jinan Fiaidhi is a Full Professor of Computer Science, Lakehead University, Ontario, Canada. She is also an Adjunct Research Professor with the University of Western Ontario, Canada. Her research interests includes: Collaborative Learning Environments, Enterprise Mashups, Web 2.0, Mobile Learning and Peer-to-peer programming.

Osama Mohammed is a final year HBEng student with the Department of Software Engineering, Lakehead University, Ontario, Canada.