Java Application Programming Interface (API) for Annotation Imaging Markup (AIM)

Hakan Bulu[1], Daniel L. Rubin[2]

[1]Dokuz Eylul University, Department of Computer Engineering, Izmir, Turkey

[2] Stanford University, Department of Radiology and Center for Biomedical Informatics Research, Stanford, U.S.

1. Introduction

About AIM

Information Sciences in Imaging at Stanford (ISIS) Laboratory of Stanford University is currently funded by the National Cancer Institute of the National Institutes of Health to develop techniques and tools to enable extracting quantitative and semantic information from radiology images in a standard format called Annotation and Image Markup (AIM).

The goal of the AIM project is to develop a mechanism, for modeling, capturing, and serializing image annotation and markup data that can be adopted as a standard by the medical imaging community. The AIM project produces both human and machine-readable artifacts. One of the most important points about the AIM is storing the annotations as XML (Extensible Markup Language) format called as “AIM XML”. Additional applications transform the AIM XML into DICOM-SR, HL7-CDA XML and AIM ontology instances. The AIM model is shown in Figure 1. For more information about the AIM project; [1], [2], [3].

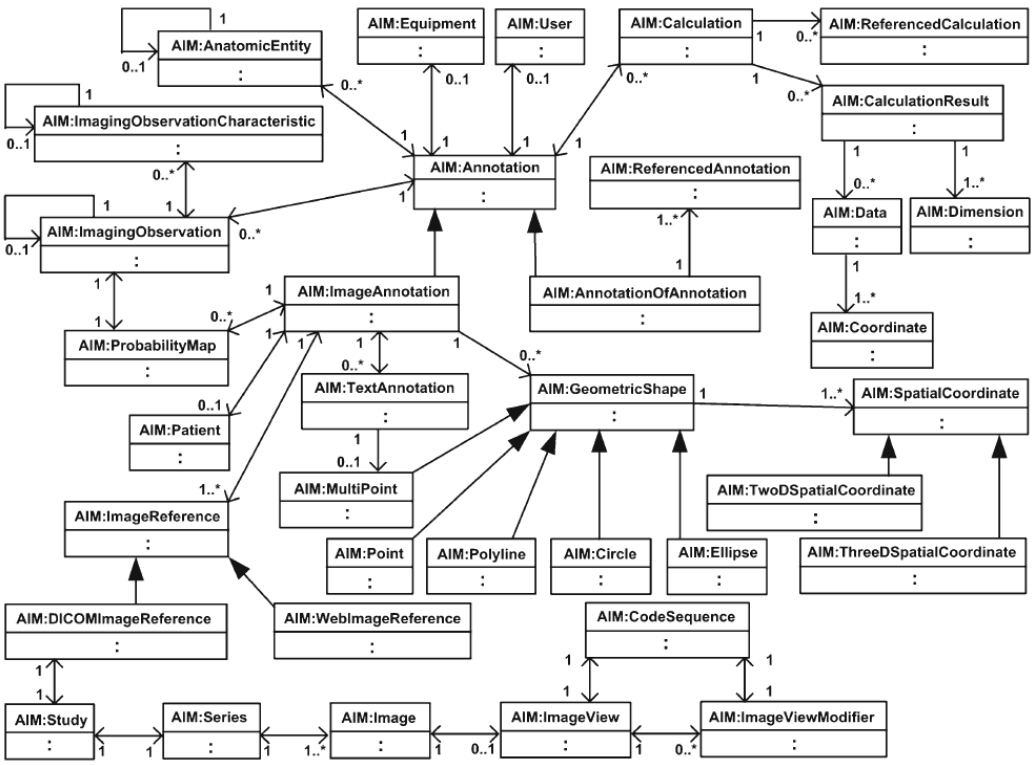

Figure 1 - AIM UML Class Diagram

Simplify the usage of AIM structure.

Structure of the AIM is not small and understanding this structure can be difficult when you become familiar. Also, if any developer/researcher wants to use AIM, they have to know how to parse and write XML files. In other words they have to know XML technologies. Because of that, to simplify the usage of AIM Structure for Java programmers, we have developed an Application Programming Interface (API) called as “AIM API”. The AIM API [4] was developed in Java programming language by using NetBeans IDE 7.0.1 and it creates an object model, using Java classes for each class in the AIM Schema. The class hierarchy of the API closely follows the AIM Schema. Each class in the object model provides Set and Get methods for every attribute in the corresponding AIM Schema class. You can download the source code of the API from [5].

Perform operations (read and write) for AIM XML files, both on hard drive (local computer) and XML Database (remote server).

By using the API, users can not only create new AIM XML files, but also edit any existing AIM XML files which can be current or older version of the AIM. The XML files can be saved to both local file system and remote XML database. Additionally, users are able to

query one or set of AIM Annotations from the server by using predefined functions. Figure xyz illustrates the

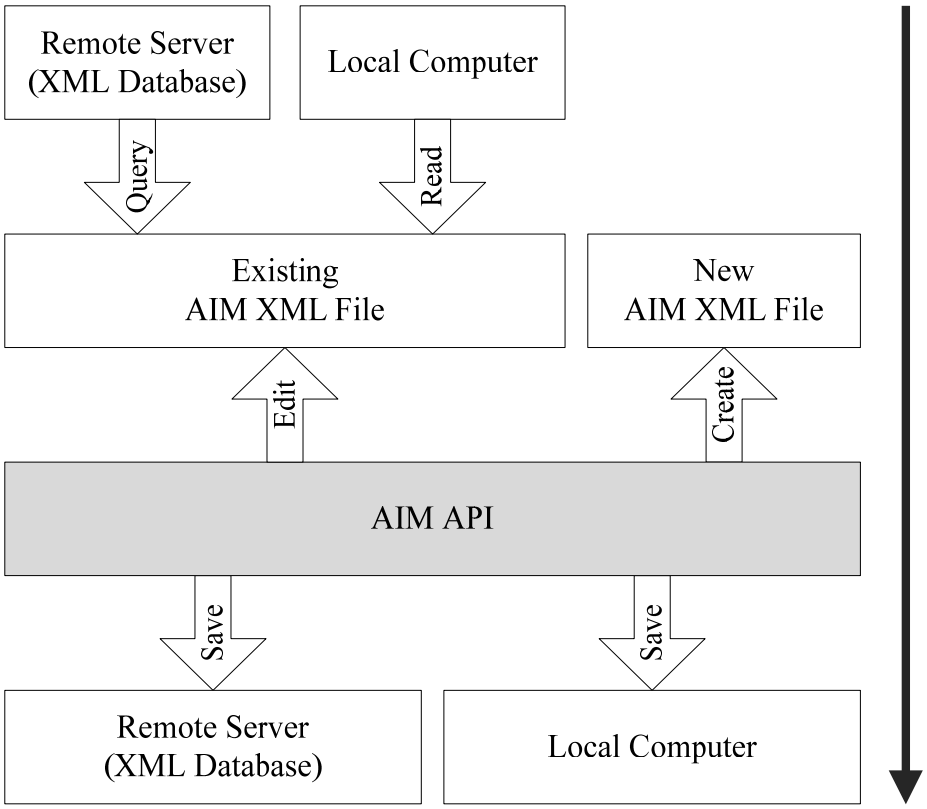

Figure xyz – General system overview of the API

While the AIM version will be increased, there is no significant change for the usage of the API.

Since the AIM project started to develop, the version of AIM XSD have been updated. So, reading/parsing any version AIM XML file is an important capability for the API. At this point

Instead of creating individual AIM API for each AIM XSD file, we have

So, the users must

To be able to analysis AIM annotations with Semantic Web Technologies, the annotations must be converted AIM Ontology Instances. So, I have updated the API with respect of this necessity. In following sections sample usage of the API is given to simulate the annotation of a lesion on the mammogram.

Validate the existing and new AIM XML files based on the AIM XML Schema (XSD). In this way, reducing inconsistency between AIM XML files.

Each AIM XML file has to be valid according to AIM XSD so validation process is very important.

2. Methods

Which programming language did we use?

We use Java programming language to develop the AIM API. Java is an Object oriented application programming language developed by Sun Microsystems. Java is a very powerful general-purpose programming language. Due to its versatility, it is a platform independent language, be it a hardware platform or any operating system.

XML Overview

XML stands for Extensible Markup Language and was defined 1998 by the World Wide Web Consortium (W3C). XML is designed to transport and store data. A XML document contains set of element; each element has a start tag, content and an end tag. A XML document must have exactly one root element, e.g. one tag which encloses the remaining tags. XML is key sensitive so it makes a difference between capital and non-capital letters. A XML file is required to be well-formatted. Well-formed XML must apply to the following conditions:

- A XML document always starts with a prolog which describes XML file. This prolog can be minimal, e.g. `<?xml version="1.0"?>`
- Every tag has a closing tag.
- All tags are completely nested.

A XML file is valid if it is well-formed and if it is contains a link to a XML schema (XSD) and is valid according to the schema.

XSD Overview

The XML Schema definition language (XSD) enables you to define the structure (elements and attributes) and data types for XML documents. A XML schema describes the coarse shape of the XML document such as what fields an element can contain, which sub elements it can contain, ordering of tags in the document etc. It can also describe the values that can be placed into any element or attribute. While, a well-formed XML document is one that satisfies

the usual rules of XML, a valid document is one that is well-formed and that satisfies a schema.

Java XML Overview

Java contains several methods to access XML. The most popular methods to parse XML files with Java are using DOM API (Document Object Model) and SAX (Simple API for XML). In DOM, user can access the XML document over an object tree. DOM can be used to read and write XML files. On the other hand, SAX provides sequential reading of XML files. SAX can only read XML documents.

3. Results

Who will use the API?

List of the projects using AIM API can be expressed…

Sample usage of the API

According to ACR BI-RADS mammography atlas each mass has three attributes; *shape*, *margin* and *density*. Furthermore, each attribute has its set of allowed values. For example, mass shape can be *round*, *lobular*, *oval* or *irregular*. Figure xyz shows one of the mass in left breast of the patient with its boundary, where the mass has *irregular* shape, *spiculated* margin and *high* density.

For each `Image Annotation (IA)` instance, we add an `Imaging Observation (IO)` to express high level features. And each `IO` instance has its own `Imaging Observation Characteristic(s) (IOC)`. The count of `IOC` instances depends on the count of the high level features of the AOA and AOC. For example, if AOA belongs to a mass, its `IO` has three `IOC` instances, one for Shape of the mass, one for Margin of the mass and one for Density of the mass. Figure xyz displays the AIM `IO` instance with its `IOC` instances for a Mass in LCC view. On the other hand for IAOC, its `IO` has only one `IOC` which expresses the Breast Density of the mammographic case.

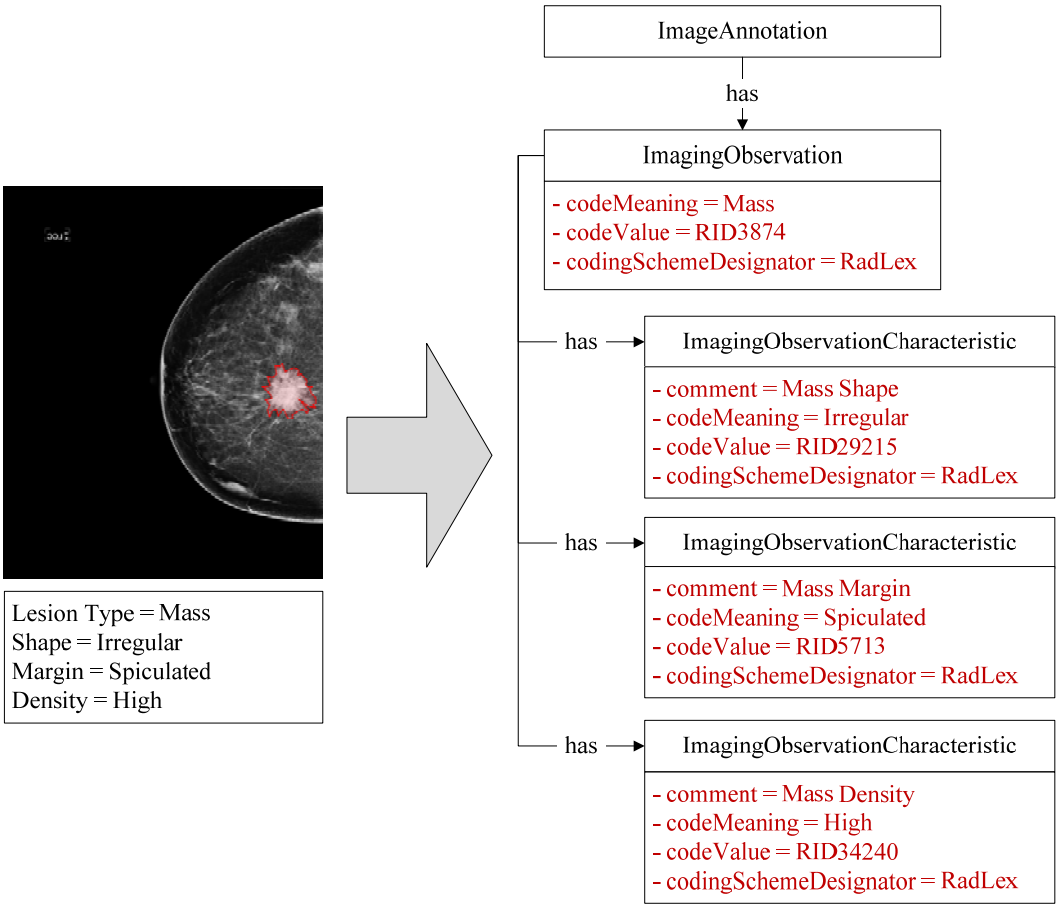

Figure xyz –

```
//*** Creating an instance of IO class (io)
ImagingObservation io;
io = new ImagingObservation();
io.setCagridId(0);
io.setCodeMeaning("Mass");
io.setCodeValue("RID3874");
io.setCodingSchemeDesignator("RadLex");

//*** Creating an instance of IOA (ioaShape) for Mass Shape
ImagingObservationCharacteristic ioaShape;
ioaShape = new ImagingObservationCharacteristic();
ioaShape.setCagridId(0);
ioaShape.setComment("Mass Shape");
ioaShape.setCodeMeaning("Irregular");
ioaShape.setCodeValue("RID29215");
ioaShape.setCodingSchemeDesignator("RadLex");

//*** Creating an instance of IOA (ioaMargin) for Mass Margin
ImagingObservationCharacteristic ioaMargin;
ioaMargin = new ImagingObservationCharacteristic();
ioaMargin.setCagridId(0);
ioaMargin.setComment("Mass Margin");
ioaMargin.setCodeMeaning("Spiculated");
ioaMargin.setCodeValue("RID5713");
ioaMargin.setCodingSchemeDesignator("RadLex");

//*** Creating an instance of IOA (ioaDensity) for Mass Density
```

```
ImagingObservationCharacteristic ioaDensity;
ioaDensity = new ImagingObservationCharacteristic();
ioaDensity.setCagridId(0);
ioaDensity.setComment("Mass Density");
ioaDensity.setCodeMeaning("High");
ioaDensity.setCodeValue("RID34240");
ioaDensity.setCodingSchemeDesignator("RadLex");

//*** Adding all instances of IOA to the IO instance
io.addImagingObservationCharacteristic(ioaShape);
io.addImagingObservationCharacteristic(ioaMargin);
io.addImagingObservationCharacteristic(ioaDensity);

//*** Adding instance of IO class to the Image Annotation
iAnnotation.addImagingObservation(io);
```

Figure xyz –

4. Conclusion and Future Works

What did we do?

What did we obtain?

What will we do?

5. Acknowledgements

This work is supported by The Scientific and Technological Research Council of Turkey (TUBITAK) under project number 107E217 and National Cancer Institute (NCI) through the cancer Biomedical Informatics Grid (caBIG) Imaging Workspace subcontract from Booz-Allen & Hamilton, Inc. 85983CBS43.

6. References

[1] Rubin DL, Mongkolwat P, Kleper V, Supekar K and Channin DS, Medical Imaging on the Semantic Web: Annotation and Image Markup. In: 2008 AAAI Spring Symposium Series, Semantic Scientific Knowledge Integration, Stanford University, 2008.

[2] Rubin DL, Supekar K, Mongkolwat P, Kleper V and Channin DS, Annotation and Image Markup: Accessing and Interoperating with the Semantic Content in Medical Imaging. IEEE Intelligent Systems 24(1): 57-65, 2009.

[3] Channin DS, Mongkolwat P, Kleper V, Sepukar K and Rubin DL, The caBIG Annotation and Image Markup Project. J Digit Imaging, 2009.

[4] AIM API Web Page, http://www.stanford.edu/group/qil/cgi-bin/mediawiki/index.php/AIM_API, 2011.

[5] AIM API Spurce Code, http://sourceforge.net/projects/aimapi/, 2011.