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].

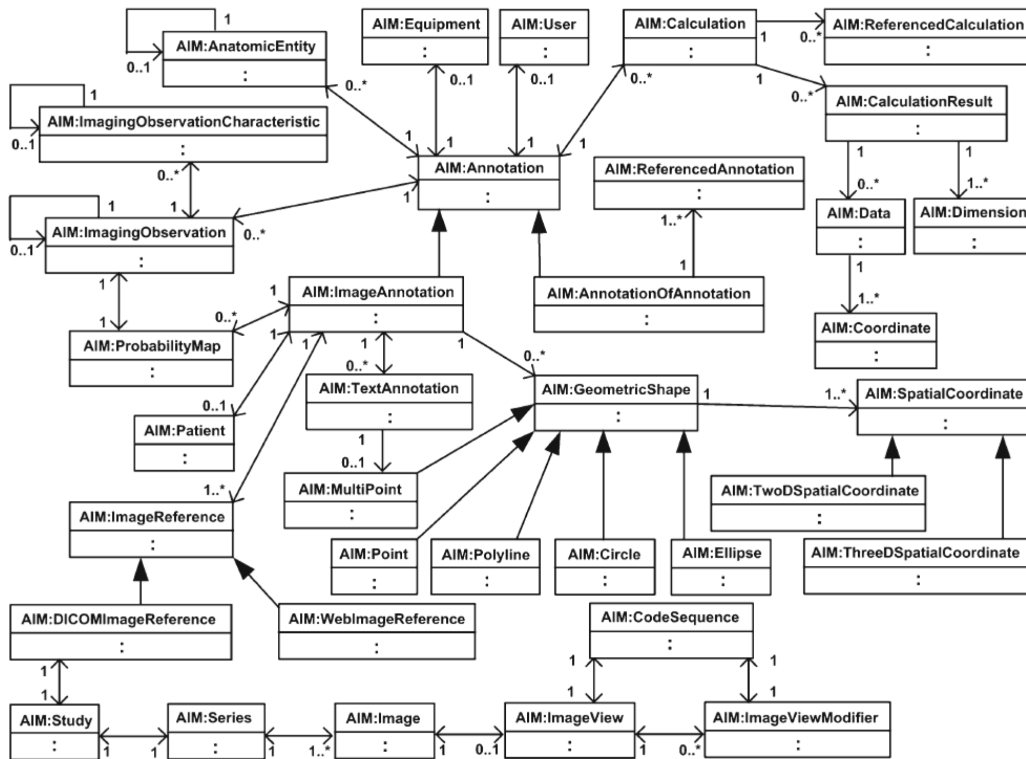

Figure 1 - AIM UML Class Diagram

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

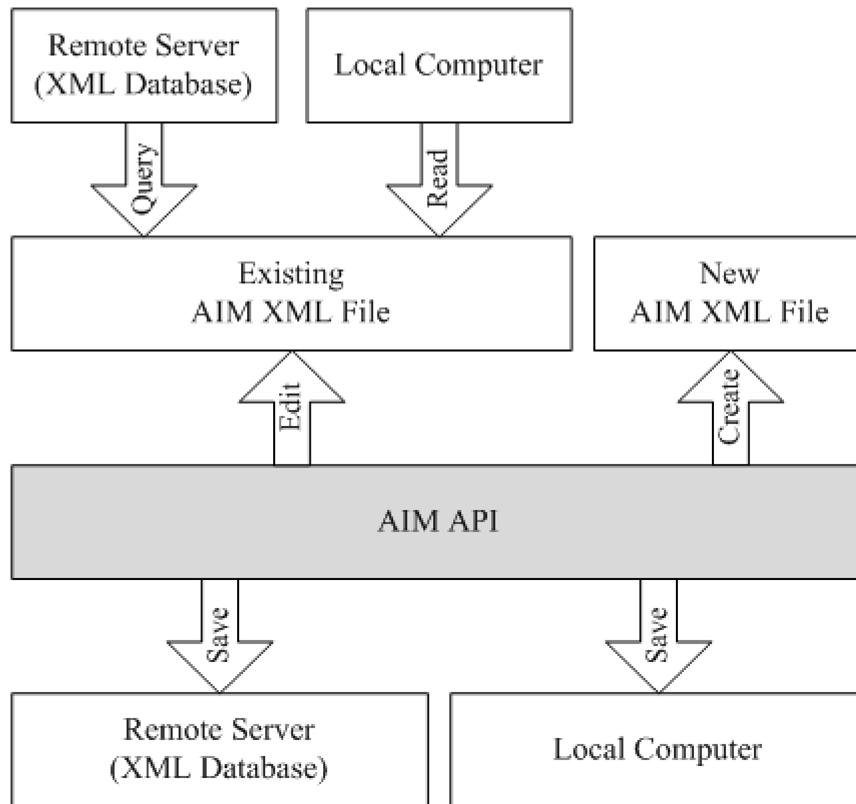

Figure 2– General system overview of the API

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 upgraded (e.g., TCGA, 3.0 and 4.0) and each of them has some differences. So, reading/parsing any version AIM XML file is an important capability for the API. Otherwise, AIM users not only have to understand/clarify the differences, but also they may have to change/update their codes for each new release of AIM XSD. Besides, in practice, source code update may increase the error occurrence percentage of AIM users' projects and get their times.

For these reasons, instead of creating individual AIM API for each AIM version, we have updated our API to be able to parse/read any version of AIM XML file. Usage of the API is same for each AIM version, which means that the API users do not need to change their codes or algorithms depend on the AIM version. The API automatically detects the version of any given AIM XML file and creates required java class instances. Then, users can read/edit the AIM XML file and save it in current AIM version.

Creation of AIM Ontology instances

On the other hand, to be able to analysis AIM annotations with Semantic Web technologies, the annotations must be converted to AIM ontology instances. So, we have updated the API with respect of this necessity. API users can save their AIM annotations into any existing or new AIM OWL (Web Ontology Language) ontology file. While Figure 3 shows syntax of a sample Person instance in AIM XML, Figure 4 shows RDF syntax of the same Person instance in an AIM OWL file.

```
<Person birthDate="1953-01-01T00:00:00" sex="F" cagridId="0"
id="P481" name="1.3.6.1.4.1.9328.0022" />
```

Figure 3 – Sample Person in AIM XML

```
<AIM:Person rdf:ID="Person_P481">
  <AIM:birthDate rdf:datatype="http://www.w3.org/2001/XMLSchema#date">
    1953-01-01
  </AIM:birthDate>
  <AIM:sex rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    F
  </AIM:sex>
  <AIM:cagridId rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
    0
  </AIM:cagridId>
  <AIM:id rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    P481
  </AIM:id>
  <AIM:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
   1.3.6.1.4.1.9328.0022
  </AIM:name>
</AIM:Person>
```

Figure 4 – Sample Person in AIM Ontology

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 to provide consistency between AIM annotations.

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

In this sections sample usage of the API is given to simulate the annotation of a lesion on the mammogram. `ImageAnnotation` class of AIM is used to annotate images and the class is inherited from `Annotation` abstract class which captures general description of the AIM annotation such as name, version etc. Figure 5 shows creation of an `ImageAnnotation` instance with its required parameters.

```
//*** Creating an ImageAnnotaion class instance (iAnnotation)
ImageAnnotation iAnnotation = new ImageAnnotation();

//*** Setting required fields of the iAnnotation
iAnnotation.setCagridId(0);
iAnnotation.setAimVersion("AIM 3.0");
iAnnotation.setDateTime("2012-01-16T15:32:14");
iAnnotation.setName("Lesion 1");
iAnnotation.setUniqueIdentifier("ID-12345");
iAnnotation.setCodeValue("0");
iAnnotation.setCodeMeaning("Mass");
iAnnotation.setCodingSchemeDesignator("DCM");
iAnnotation.setComment("Sample Usage of AIM API");
```

Figure 5 – Creation of an AIM Image Annoation

Each `ImageAnnotation` instance must have a `Person` instance to state owner (Patient) of the annotation. The *CagridId* is used to uniquely identify an instance of a class within a system implementing caGRID technology [6], *name* and *id* attributes of `Person` class must have been set. In addition to these attributes `Person` class may have *birthDate*, *sex* and *ethnicGroup*. Figure 6 shows creation of a sample `Person` class by using AIM API and it is added to its `ImageAnnotation` instance.

```
//*** Creating a Person instance (person)
Person person = new Person();
person.setCagridId(0);
person.setName("Name Surname");
person.setId("112233");
person.setSex("F");
person.setBirthDate("1965-02-12");

//*** Adding Person to ImageAnnotation
iAnnotation.addPerson(person);
```

Figure 6 - Creation of an AIM Person

The image references classes specify the image or collection of images being annotated. AIM annotations may reference DICOM images or web images. In this paper we just give usage of `DICOMImageReference` class which simulates a subset of the DICOM information model. The class has one `ImageStudy` instance that has one or more `ImageSeries` instances which in turn have one or more `Image` instances. `ImageStudy` instance has *CagridId*, *UID* and *Date* and *Time*. `ImageSeries` instance has *CagridId* and *UID*. `Image` instance has *CagridId*, *SOP class UID*, *SOP instance UID*.

```
//*** Creating a DICOMImageReference instance (dicomImageReference)
DICOMImageReference dicomImageReference = new DICOMImageReference();
dicomImageReference.setCagridId(0);

//*** Creating a ImageStudy instance (imageStudy)
ImageStudy imageStudy = new ImageStudy();
imageStudy.setCagridId(0);
imageStudy.setInstanceUID("LCC");
imageStudy.setStartDate("2012-01-16");
imageStudy.setStartTime("12:45:34");

//*** Creating a ImageSeries instance (imageSeries)
ImageSeries imageSeries = new ImageSeries();
imageSeries.setCagridId(0);
imageSeries.setInstanceUID("LCC");

//*** Creating a Image instance (image)
Image image = new Image();
image.setCagridId(0);
image.setSopClassUID("112233");
image.setSopInstanceUID("445566");

//*** Adding Image to ImageSeries
imageSeries.addImage(image);
//*** Adding ImageSeries to ImageStudy
imageStudy.setImageSeries(imageSeries);
//*** Adding ImageStudy to ImageReference
dicomImageReference.setImageStudy(imageStudy);
//*** Adding ImageReference to ImageAnnotation
iAnnotation.addImageReference(dicomImageReference);
```

Figure 7 - Adding DICOM Image's information to the Image Annotation

`ImagingObservation` class is used for the description of “things” in the image. For example “mass,” “calcification,” etc. `ImagingObservation` instance can be associated with one or more `ImagingObservationCharacteristic` instance which descriptors of `ImagingObservation`.

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 8 shows a mass has *irregular* shape, *spiculated* margin and *high* density in the left breast of the patient with its boundary and representation of the mass in AIM structure. Additionally, Figure 9 shows usage of AIM API to add the annotation of the mass into AIM Image Annotation.

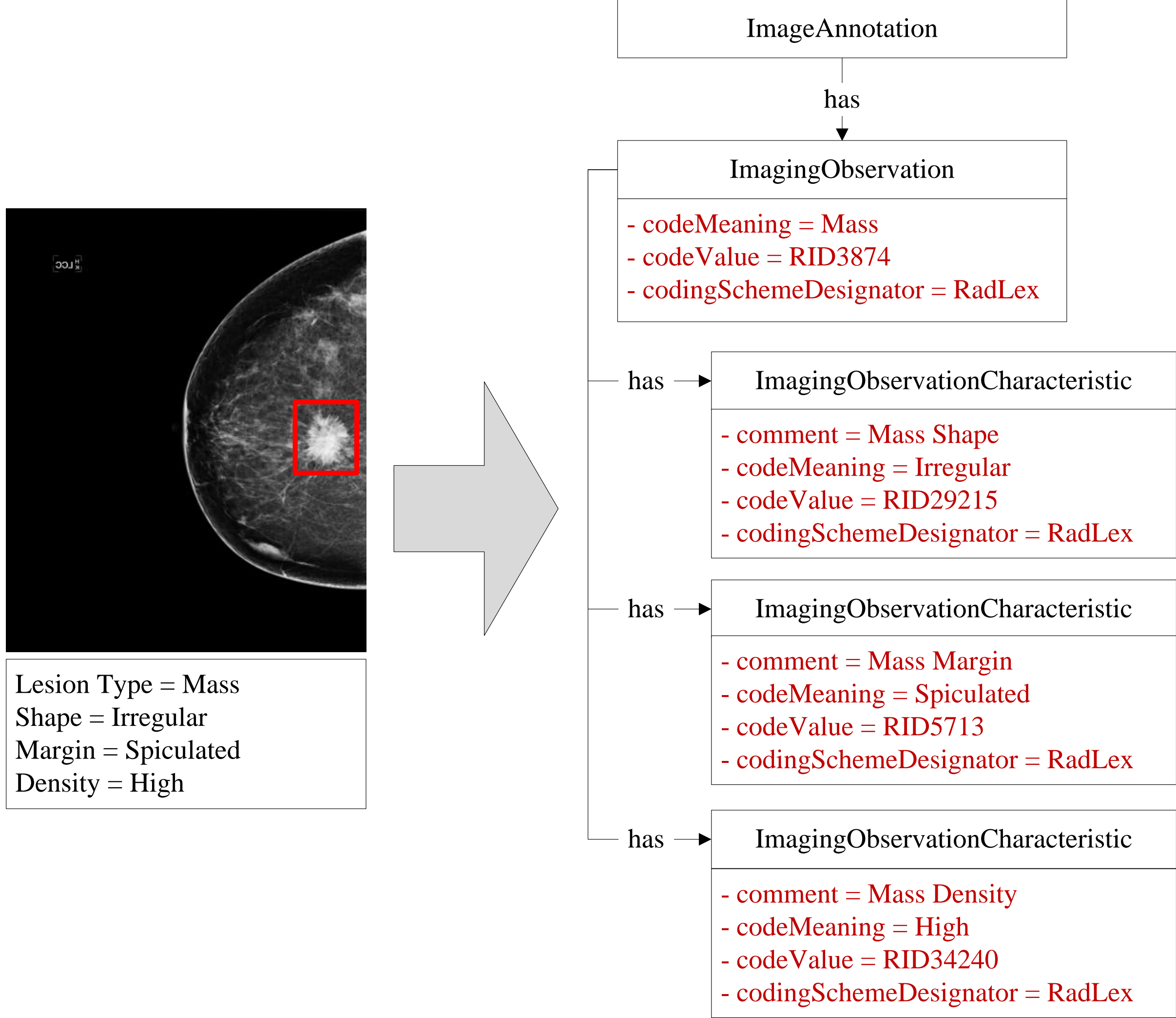

Figure 8 – Sample Breast Lesion and its Representation in AIM

```
//*** 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 9 - Creation of an AIM Imaging Observation

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.

[6] The caGrid, https://cabig.nci.nih.gov/community/workspaces/Architecture/caGrid, 2012