SIEMENS

Algorithm Validation Toolkit
AVT2EXT

Algorithm Validation Toolkit
AVT2EXT

Design Specification R1.0

Last Change: May 10, 2010

Robert W. Schwanke
Fabian Moerchen
Philipp Hussels
Jie Zheng
Lawrence Tarbox[1]

[1] Washington University, St. Louis, MO.

History

Document History

Version/ Status	Date of Issue	Author	Change and Reason of Change
0.1 Draft		Robert W. Schwanke	First draft
0.2 Revision		Fabian Moerchen Philipp Hussels	Revision adding AD content
0.3 Revision		Jie Zheng	Revision adding IA, MVT, AE contents
0.4		Robert W. Schwanke	Final draft prior to AVT2EXT release
1.0	5-May-2010	Robert W. Schwanke	AVT2EXT Final Delivery

History of released Versions

Version	Release date	Product Version
1.0	5-May-2010	Robert W. Schwanke

Table of Contents

History 2
Table of Contents 3
1. Introduction 5
  1.1. Purpose of the document 5
  1.2. Area of validity of the document 5
  1.3. Definitions and abbreviations 5
  1.4. References 5
2. General description 5
  2.1. Overview 5
  2.2. Required functionality 5
  2.3. Interfaces and interactions 5
  2.4. Dependencies 7
  2.5. Risk analysis 7
  2.6. Development environment 7
3. Functional design topics 7
  3.1. Module breakdown 7
  3.2. Image Reader design 8
  3.3. Algorithm Execution design 13
  3.4. Measurement Variability Tool Design 15
  3.5. AD Core design 20
    3.5.1. AD API 20
    3.5.2. AD Implementation 21
  3.6. AD Installer design 22
  3.7. XIP™ design 24
    3.7.1. XIP Host™ 25
    3.7.2. XIP Libraries™ 26
    3.7.3. XIP™ Application Support Java Classes 26
  3.8. Safety 27
  3.9. Dynamic behavior 27
  3.10. Interface Topics 27
    3.10.1. Image Reader UI State Transition design 27
    3.10.2. Hardware interfaces 28
    3.10.3. Operating system interfaces 28
    3.10.4. External software interfaces 28
    3.10.5. R Library Interfaces 28
    3.10.6. XIP Libraries™ Interfaces 29
  3.11. Internal software interfaces 29
  3.12. Data dictionary 30
4. Data design topics 30
  4.1. AIM design for CDRH case study 30
    4.1.1. Seeds 30
    4.1.2. RECIST 30
    4.1.3. WHO 30
    4.1.4. volume 30
    4.1.5. work-arounds 31
    4.1.6. variations 31
  4.2. AIM design for TCGA case study 31
  4.3. Database design 31

5. API Documentation 31
5.1. Application APIs 31
5.2. AD APIs 31
5.3. XIPHost_AVT 31
6. General quality features 31
6.1. MISC_operational_pilot 31
6.2. MVT_CDRH_case_capacity, MVT_CDRH_annotation_capacity 32
6.3. Availability 32
6.4. User friendliness 32
6.5. Maintainability 32
6.6. Transferability, portability 32
6.7. Testability 32
7. Unsettled points / open issues 32

1. Introduction

1.1. Purpose of the document

The purpose of this document is to give an overview of the internal design of the software and to explain several significant design solutions generated during the development of AVT2EXT. It is not intended as comprehensive design documentation, since most of the design is adequately apparent in the code itself.

1.2. Area of validity of the document

Algorithm Validation Toolkit, as described in the accompanying Requirement Specification and Functional Specification. This is version AVT2EXT.

1.3. Definitions and abbreviations

See *AVT2EXT Definitions and Abbreviations,* a separate document.

1.4. References

[1] AVT2EXT Administration Guide , version 1.0
[2] AVT2EXT Definitions and Abbreviations, version 1.0
[3] AVT2EXT Design Specification, version 1.0
[2] AVT2EXT Functional Specification, version 1.0
[4] AVT2EXT Image Annotation User Manual, version 1.0
[5] AVT2EXT Implementation Plan, version 1.0
[6] AVT2EXT Installation Guide, version 1.0
[7] AVT2EXT Measurement Variability Tool User Manual, version 1.0
[8] AVT2EXT Programming Guide, version 1.0
[9] AVT2EXT Requirement Specification, version 1.0
[10] AVT2EXT Vision, Scope, and Technical Overview, version 1.0
[11] www.openxip.org

2. General description

2.1. Overview

See *AVT2EXT Vision, Scope, and Technical Overview*, a separate document.

2.2. Required functionality

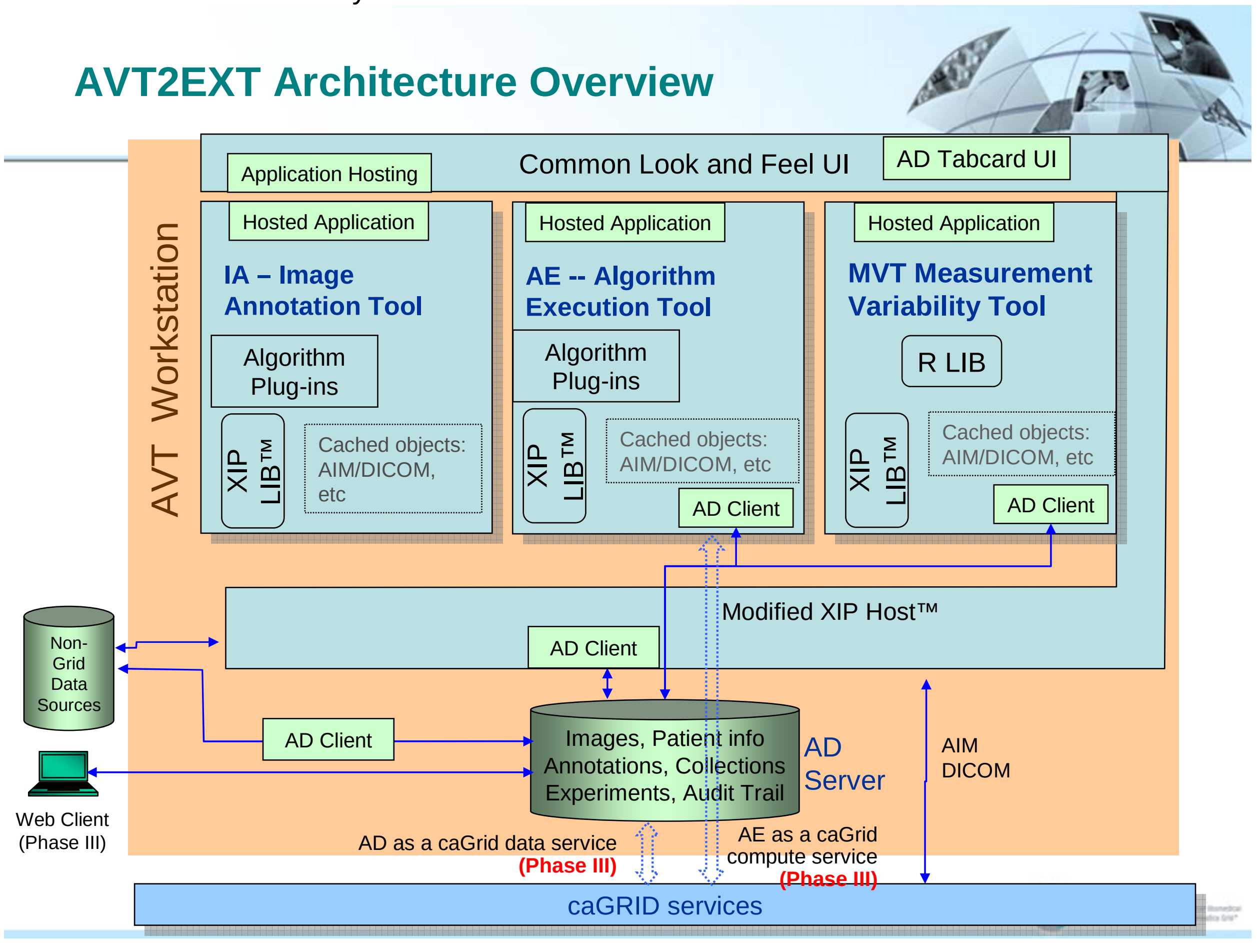
See *AVT2EXT Functional Specification*, a separate document.

2.3. Interfaces and interactions

The following diagram depicts the major components of AVT and the interfaces among them and with the external environment.

- The AVT Workstation communicates with caGrid, with non-Grid data sources, and, in the future, with web clients.
- Currently, the only caGrid connection is between XIP Host™ and caGrid data services, principally NBIA and AIME (the AIM data service at Emory University).

- In the future, the Annotation Database is intended to become a type of caGrid data service, and the Algorithm Execution component is intended to become a type of caGrid computation service.
- XIP Host™ can load and store using non-Grid data sources as well as AD and caGrid data sources.
- AD also provides a loader that can be embedded in ad hoc scripts to load data from non-Grid sources directly into AD.


AVT2EXT Architecture Overview
AVT Workstation
Application Hosting
Common Look and Feel UI
AD Tabcard UI
Hosted Application
IA – Image Annotation Tool
Algorithm Plug-ins
XIP LIB™
Cached objects: AIM/DICOM, etc
Hosted Application
AE -- Algorithm Execution Tool
Algorithm Plug-ins
XIP LIB™
Cached objects: AIM/DICOM, etc
AD Client
Hosted Application
MVT Measurement Variability Tool
R LIB
XIP LIB™
Cached objects: AIM/DICOM, etc
AD Client
Modified XIP Host™
AD Client
Non-Grid Data Sources
AD Client
Images, Patient info Annotations, Collections Experiments, Audit Trail
AD Server
AIM DICOM
Web Client (Phase III)
AD as a caGrid data service (Phase III)
AE as a caGrid compute service (Phase III)
caGRID services

- The three AVT applications interact with XIP Host™ as Hosted Applications through the DICOM Supplement 118 Application Hosting interface.
- XIP Host™ has been adapted to AVT, providing a tabcard user interface for querying and retrieving data from AD, which it passes to the hosted applications. Results returned from these applications are typically stored in AD.
- IA received DICOM Series from XIP Host™ and returns AIM/XML annotation objects, sometimes with DICOM Segmentation objects attached by reference.
- AE receives AIM/XML Seed objects and RECIST objects, retrieves the reference Series from AD, and returns AIM volume annotations.
- MVT receives AIM/XML annotations (RECIST, WHO, and Volume), looks up the referenced series if needed for visualization or metadata, and produces a variety of statistical analyses as user interface output, which can also be saved to files in various public formats.

A typical experiment proceeds as follows:

1. A curator collects the images to be marked up.

2. The curator uses IA to create seed AIMs indicating the locations of specific tumors.
3. A developer tailors IA for the specific markup procedure being studied.
4. An experimenter assigns reading tasks to different readers.
5. Each reader performs the reading tasks assigned, receiving a seed AIM as part of each task description and producing a set of AIMs (Volume, RECIST, and/or WHO).
6. The developer embeds a segmentation algorithm in a Scene Graph and installs the scene graph in AE.
7. The experimenter feeds the same seeds used in manual experiments to AE, which runs the segmentation algorithm on the same tumors, producing Volume AIMs.
8. A statistician loads the AIMs produced by manual markup and by AE into MVT, explores the results, and tells the developer what additional analyses are needed.
9. The developer extends MVT with additional analyses, typically written in the R statistics programming language.
10. The experimenter and the statistician complete their analysis of the data.

2.4. Dependencies

Module	*Depends on*
MVT	AD-core, Hosted Application interface
AE	AD-core, Hosted Application interface
IA	Hosted Application interface
AD-installer	AD-core
AD-core	DB2
XIPHost_AVT	XIP Host™, AD-core, Application Hosting interface

Note: the applications and the application framework do not directly depend on each other, only on their respective halves of the DICOM Supplement 118 application hosting standard.

2.5. Risk analysis

N/A. This product has no safety-critical aspects.

2.6. Development environment

See *AVT2EXT Implementation Plan*, a separate document.

3. Functional design topics

3.1. Module breakdown

AVT is composed of the following subsystems:

- Image Reader, currently in two variants
  - IA (Image Annotation)
  - TCGA (Specialized Image Annotation tool for TCGA Radiology project)
- Algorithm Execution (AE)
- Measurement Variability Tool (MVT)
- AD-Core (core of AD that is used in all components)

- AD-installer (AD utility that is only used to set up database and load data)
- XIPHost_AVT (AVT extensions to the XIP Host™, implementing its custom tabcard, its database interface, and special handling of combinations of AIM and DICOM objects.)

3.2. Image Reader design

The Image Reader works as an XIP™ application to provide a set of tools to view and annotate DICOM images. In AVT Phase II Extension, the Image Annotation (IA) tool is developed to load the CT series images and markup the 3D tumors. The XIP Host™ feeds the DICOM images to IA, and IA sends the annotation results as AIM xml format back to Host via WG-23 interface.

The IA consists of the Java GUI and the XIP™ scene graph implementation. The Java Swing library is used to build up the IA GUI. The XIP™ scene graph works as the core part of IA to process the DICOM images. The Java GUI interacts with the XIP scene graph via the XIP Libraries™ class "ivCanvas".

The IA XIP™ scene graph shown in Figure 1 contains all main functions to load, view, annotate, and review DICOM images/AIMs in the annotation process.

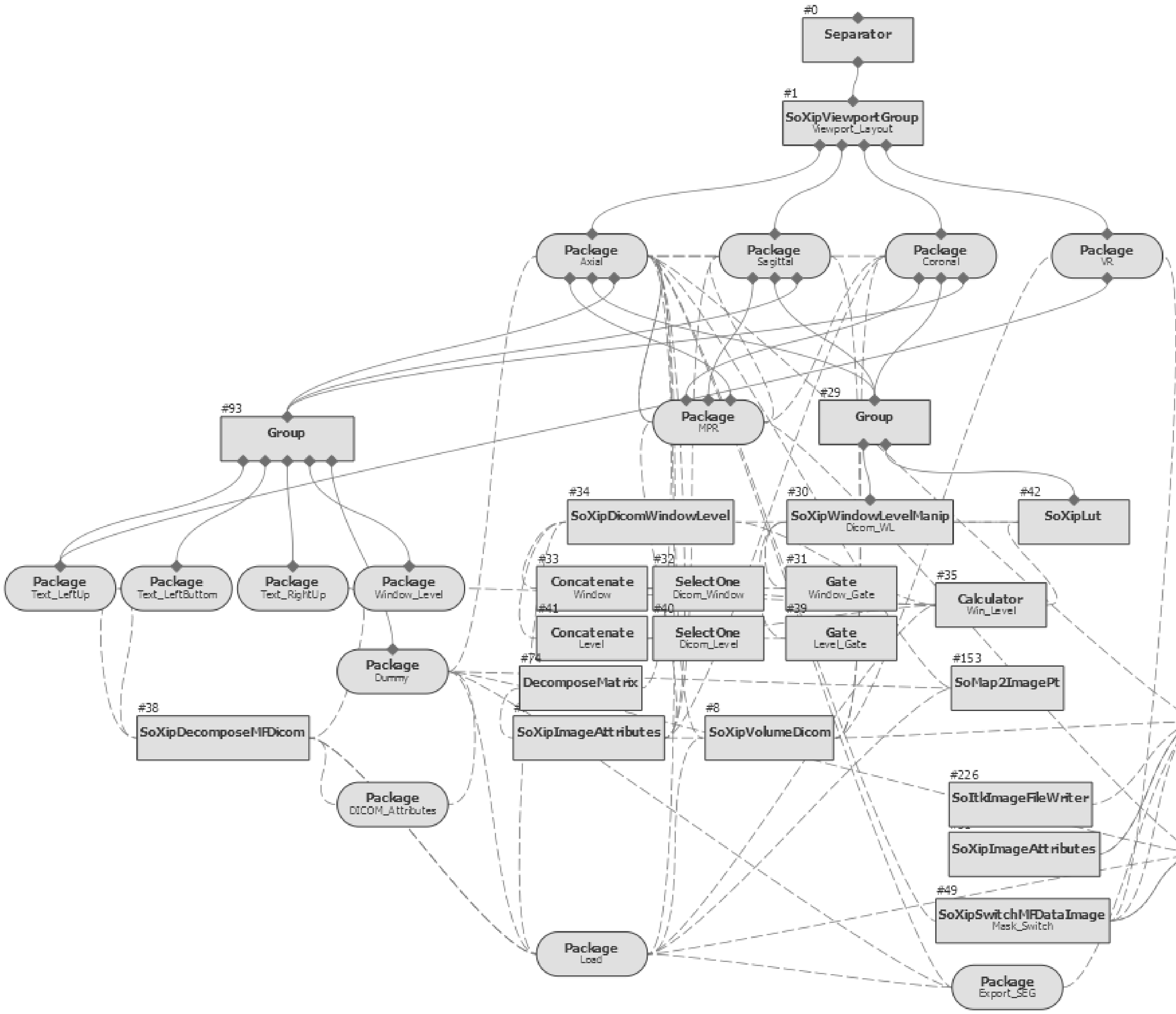


Figure 1 IA XIP scene graph

The major annotation functions, such as RECIST, WHO, and hand-free drawing, are implemented in the Package "Axial" shown in Figure 2.

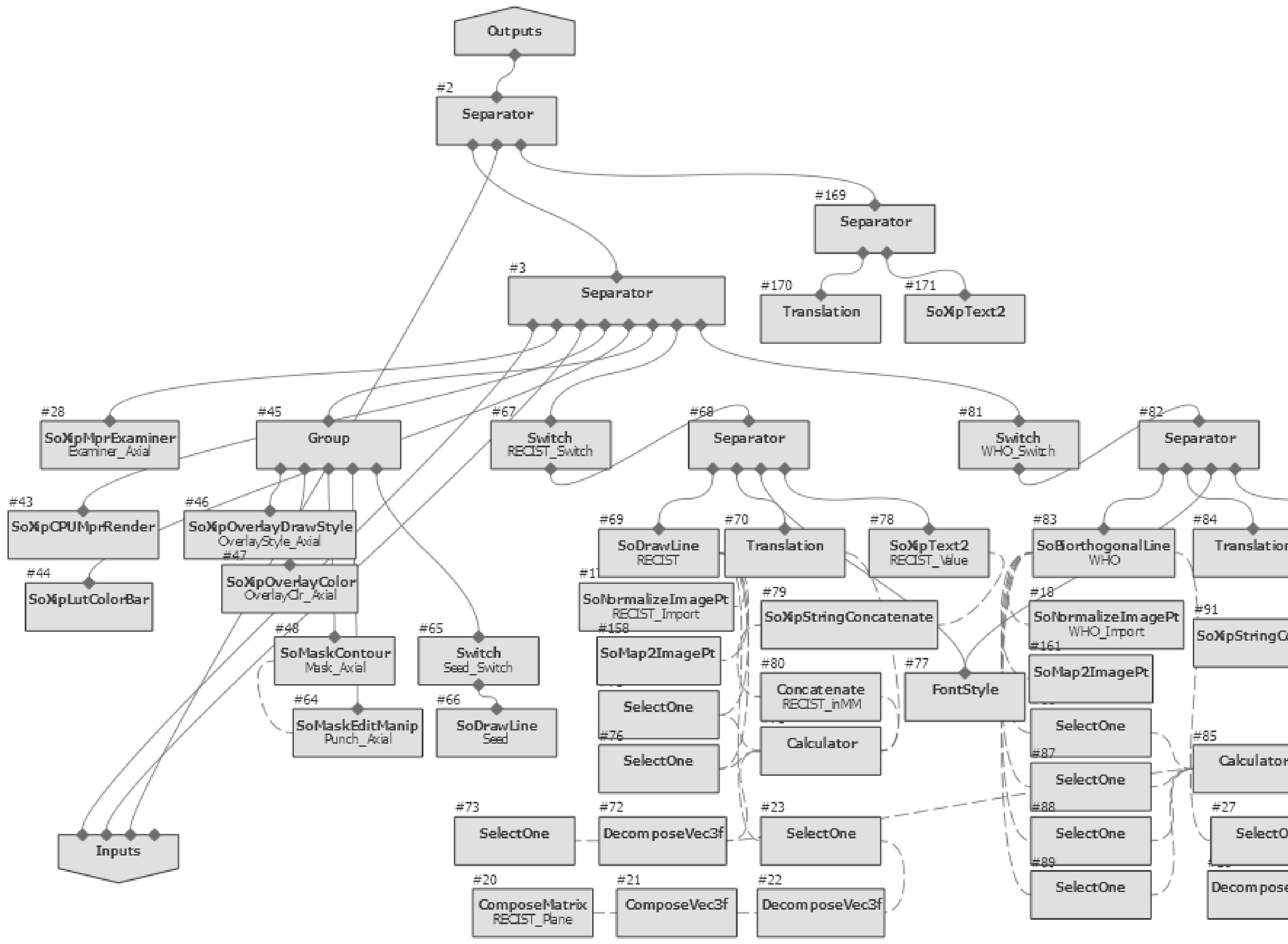


Figure 2 IA annotation scene graph

The ITK based segmentation algorithm used to generate the initial annotation is shown in Figure 3.

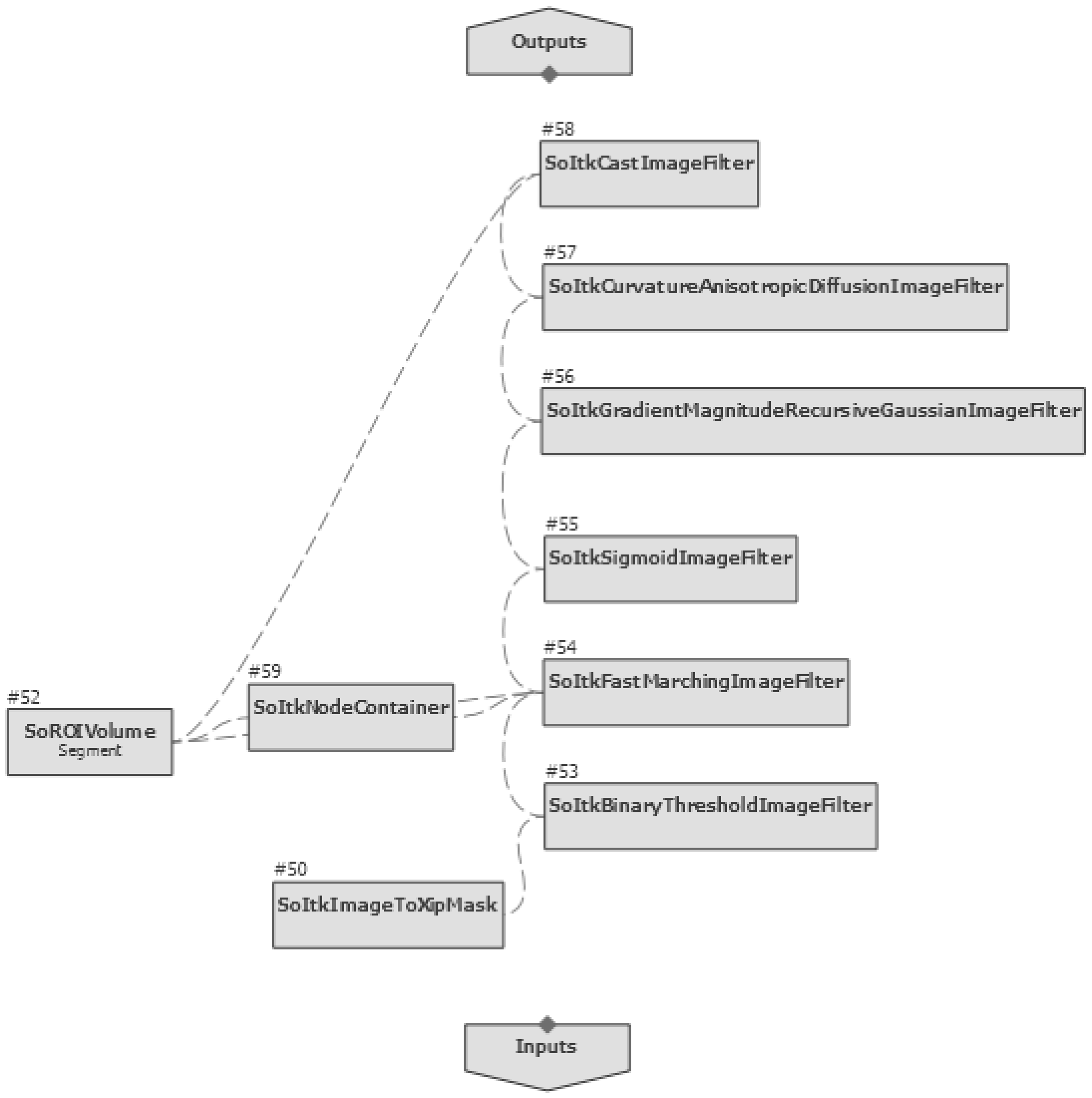

Outputs
#58
SoItkCastImageFilter
#57
SoItkCurvatureAnisotropicDiffusionImageFilter
#56
SoItkGradientMagnitudeRecursiveGaussianImageFilter
#55
SoItkSigmoidImageFilter
#54
SoItkFastMarchingImageFilter
#53
SoItkBinaryThresholdImageFilter
#52
SoROIVolume
Segment
#59
SoItkNodeContainer
#50
SoItkImageToXipMask
Inputs

Figure 3 IA ITK segmentation scene graph

The GPU based volume rendering method is used to render the 3D volume image shown in Figure 4.

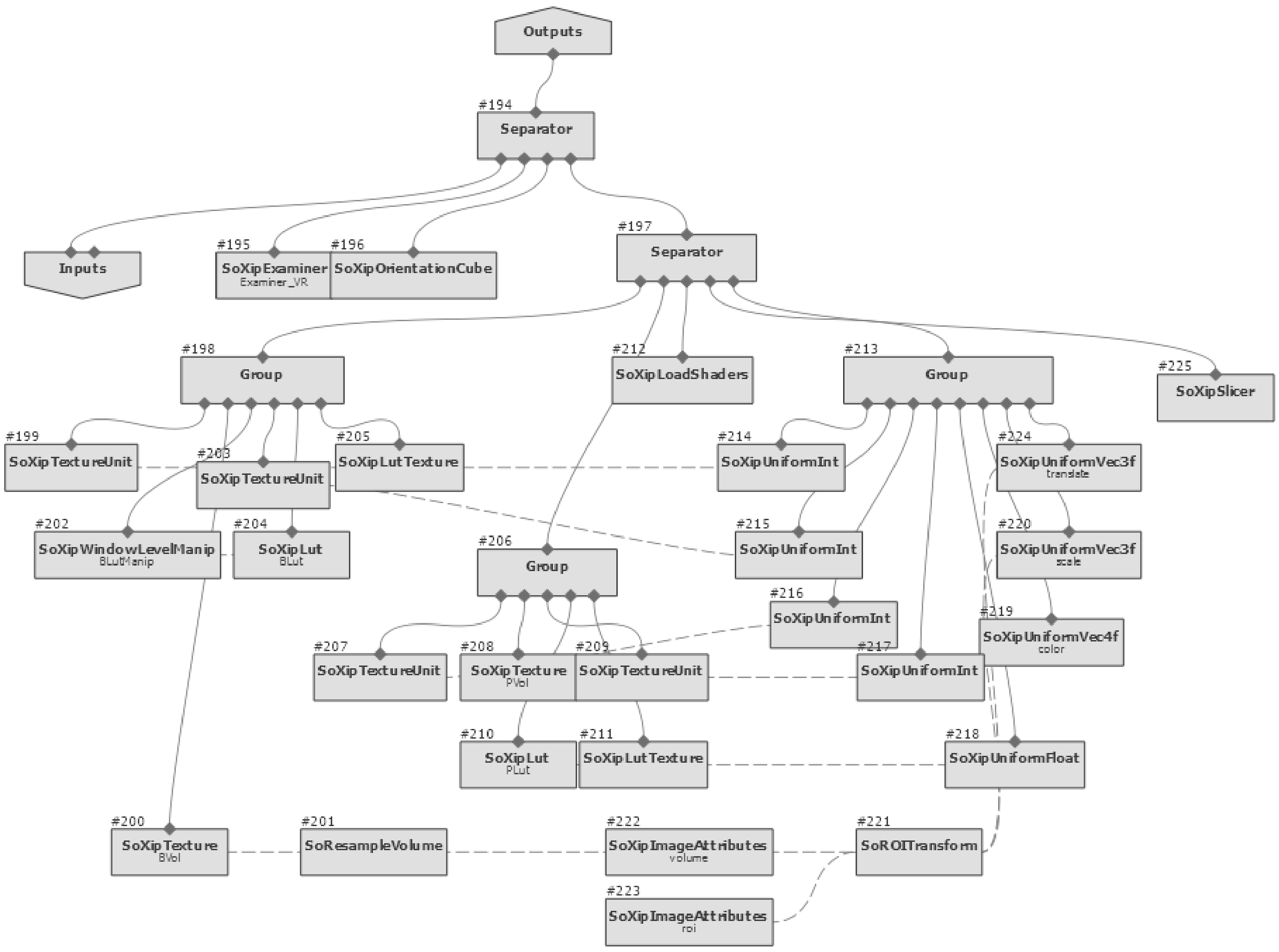


Figure 4 IA volume rendering scene graph

The DICOM images loading and pre-processing functions are shown in Figure 5.

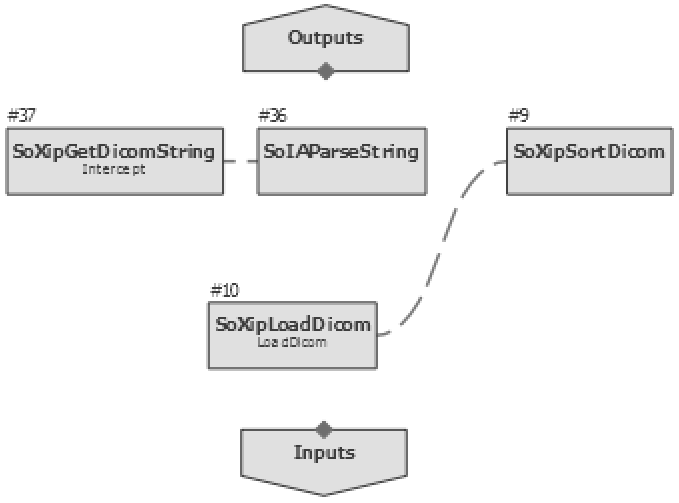

Outputs
#37 SoXipGetDicomString Intercept
#36 SoIAParseString
#9 SoXipSortDicom
#10 SoXipLoadDicom LoadDicom
Inputs

Figure 5 IA DICOM image process scene graph

The annotation results import and export functions are shown in Figure 6.

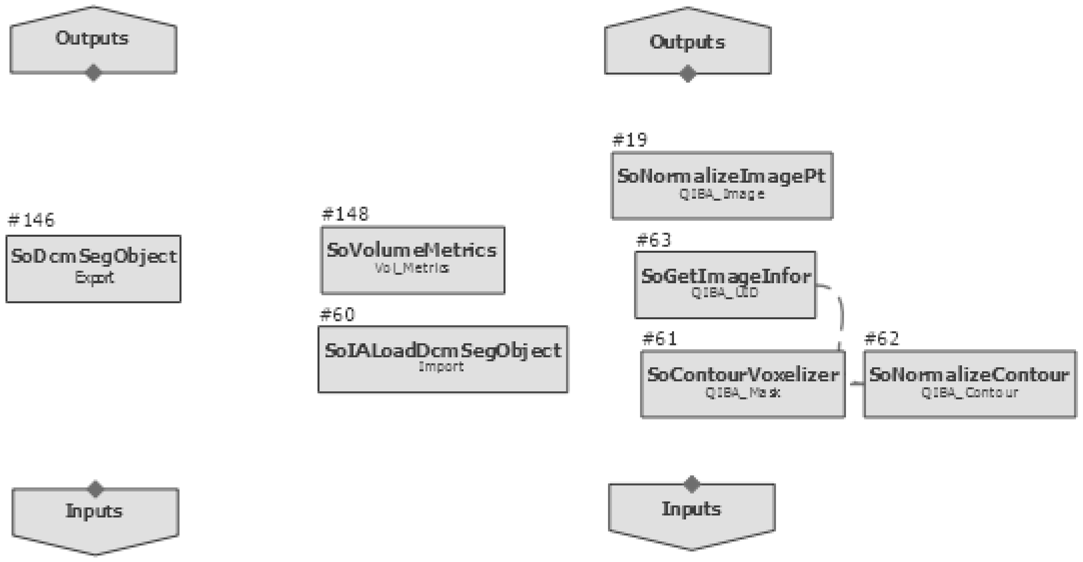


Figure 6 IA annotation results import/export scene graph

Four types of AIMs are generated in IA application, including:
1. Seed AIM
    .To store the stroke location used in the segmentation algorithm;

2. RECIST AIM
    . To store the RECIST measurement;

3. WHO AIM
    .To store the WHO measurement;

4. Volume AIM
    . To store the annotated tumor volume mask, and refer to a DICOM segmentation object;

3.3. Algorithm Execution design

The Algorithm Execution (AE) works as an XIP™ Application to run segmentation algorithms. It automatically generates AIM annotations for a selected set of images, based on a list of pre-defined seed points.

The command line mode of XIP Builder™ is used to perform the execution batch processing. The segmentation algorithm is designed as an XIP™ scene graph module.

The workflow is as follows:
1. User queries for the Series and Seeds that are associated with each Series.
2. Via the DICOM Application Hosting interfaces, the XIP Host™ passes seeds, *but not series*, to AE.

3. AE sets up the batch processing, via the XIP Builder™ functionality, to run the segmentation algorithm on the seeds and series
    a. ON CANCEL exit the batch processing.
4. Summarize batch execution.
5. Return from AE via the DICOM Application Hosting interfaces, storing AIM objects in AD or wherever XIPHost directs.

The AE scene graph is shown in Figure 7.

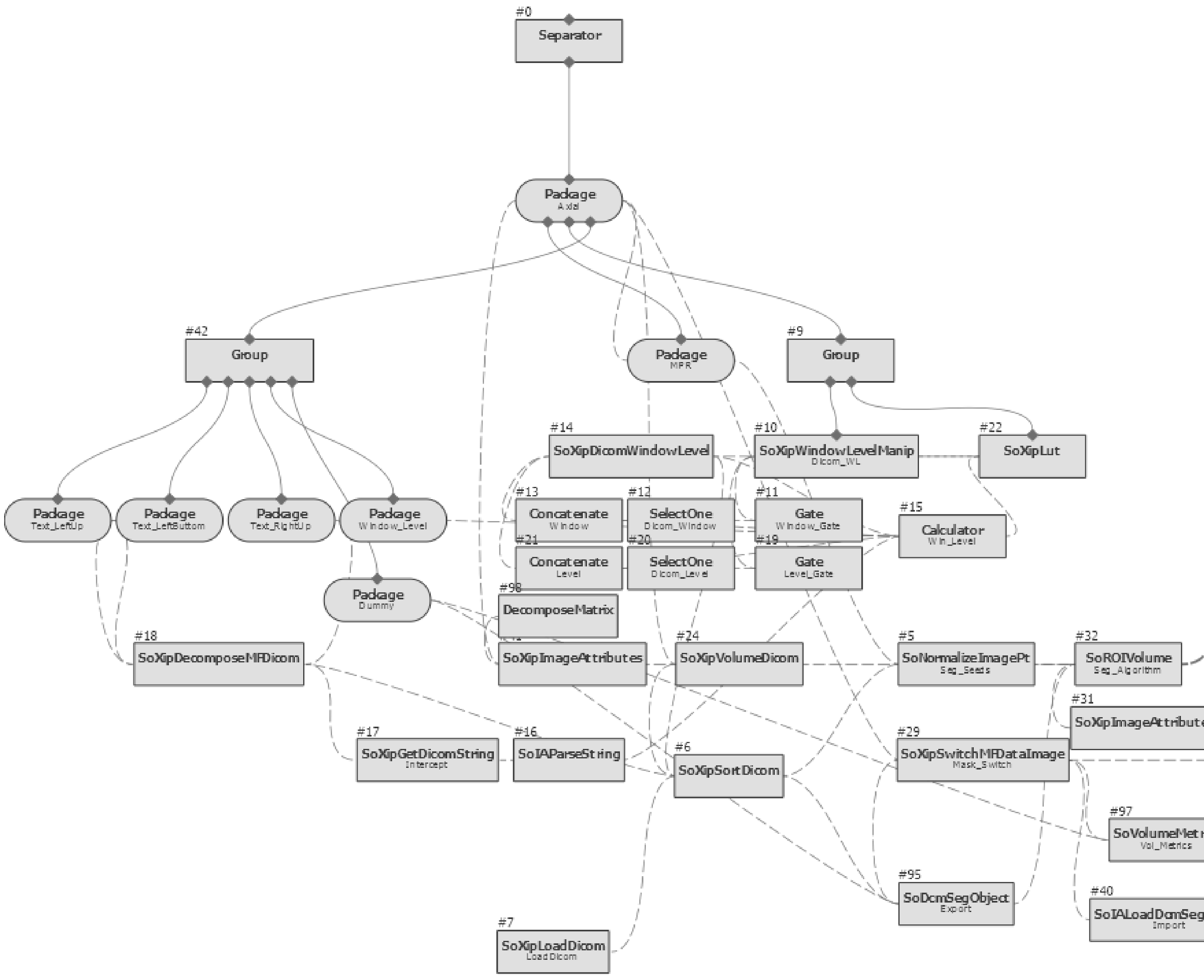


Figure 7 AE XIP scene graph

In the AE implementation, the same ITK based segmentation algorithm that is used in IA performs the segmentation. The algorithm scene graph is shown in Figure 8.

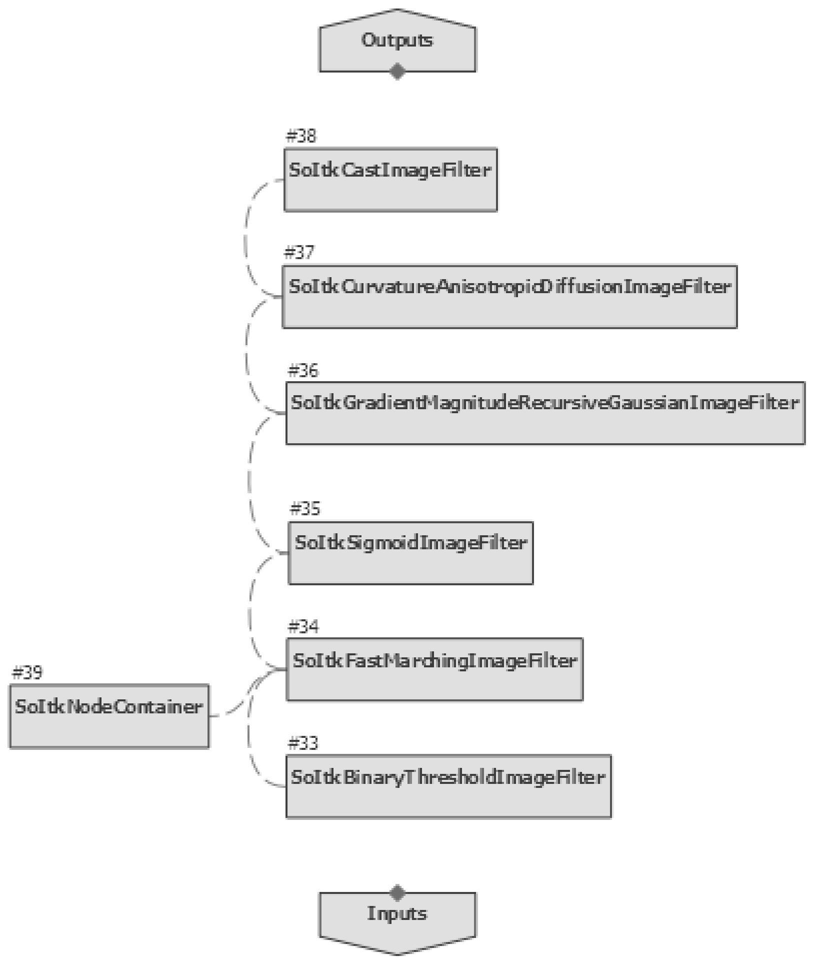

Outputs
#38
SoItkCastImageFilter
#37
SoItkCurvatureAnisotropicDiffusionImageFilter
#36
SoItkGradientMagnitudeRecursiveGaussianImageFilter
#35
SoItkSigmoidImageFilter
#34
SoItkFastMarchingImageFilter
#39
SoItkNodeContainer
#33
SoItkBinaryThresholdImageFilter
Inputs

Figure 8 AE segmentation algorithm scene graph

The AE can take two kinds of AIMs as the segmentation algorithm seed input.
1. Seed AIM
2. RECIST AIM

The Volume AIM with the DICOM segmentation object is generated in AE, and stored in XIP Host via WG-23 interface.

3.4. Measurement Variability Tool Design

The Measurement Variability Tool (MVT) works as an XIP™ application to compare annotations, and support statistical analysis such as independent variables, analysis of inter-reader and intra-reader variations, and drilling down to visualize annotation outliers overlaid on their associated images.

The R package, which is a free software environment for statistical computing and graphics, is used to perform the statistical analysis.

The workflow is as follows:

PART 1 - Choose Cases

The user performs a query on the XIP Host™ to select cases to study.

- The query can specify values for selected attributes supported by AD that are present in the Thoracic Phantom data, whether in DICOM or AIM.
- The query results are displayed in MVT in a table that includes a column of check boxes, initially checked, where the user can un-check cases that s/he does not want to use for the current analysis.

PART 2 - Choose Analysis Type

Identify analysis types

- Performance/Source of Variation - to compare a set of experimental markups with the nominal ground truth
- Reader Variability - to compare the annotation results generated by different readers and by the same reader at different time points

Identifying nominal ground truth

- The user only needs to assign the nominal ground truth if one or more of the chosen measures are error measures.
- The nominal ground truth is identified by the User Name attribute.

There is no restriction to the number of readers supported or the number of times the reader reads the same image.
The error measures will be calculated between nominal ground truth and each annotation. An annotation selected as ground truth should not be included in the results.

Part 3 - Choose measures

Each measure that one can imagine could be calculated in

- Raw value (e.g. RECIST, WHO, or numeric volume)
- Difference from nominated ground truth

User runs the measurement calculation by clicking the "RUN" button.

- Compute the selected measurements based on the comparison type.
- By default, the mean and SD are also calculated for each measurement.

Part 4 - Choose analysis statistics

- Mean
- SD
- Factorial ANOVA
- Multiple regression
- Custom script

Statistical analysis process

- Designate one of the readers as possessing absolute truth, extract his/her annotations from the population and treat them as Nominal Ground Truth segmentations.
- Select multiple error measures for analysis, including scalar volume, surface distance, and volume overlap.
- Estimate the contribution of each independent variable to the error variation of a sample population (by multiple regression with covariates of the sources of error [scalar volume, surface distance, and volume overlap] on bias)
- Implication:

a) UI provides selection boxes to choose the independent variables for ANOVA analysis.
b) No need in this case study to select ranges of values for a particular independent variable.

Visualization of the statistical results

- Bland-Altmann, scatter, and histogram charts of scalar volume, surface distance, and volume overlap on the differences between estimated measure and Nominal Ground Truth

Part 5 - Choose outlier criteria

- Cutoff range – for the individual measurement

Part 6 - Configure plots

- Scatter chart
- Bland-Altmann chart
- Histogram plot

Drilling Down

User clicks one row of the measurements table to drill down the images of the associated annotation results along with the assigned nominated ground truth.

The measurements calculation is implemented in the XIP™ scene graph shown in Figure 9.

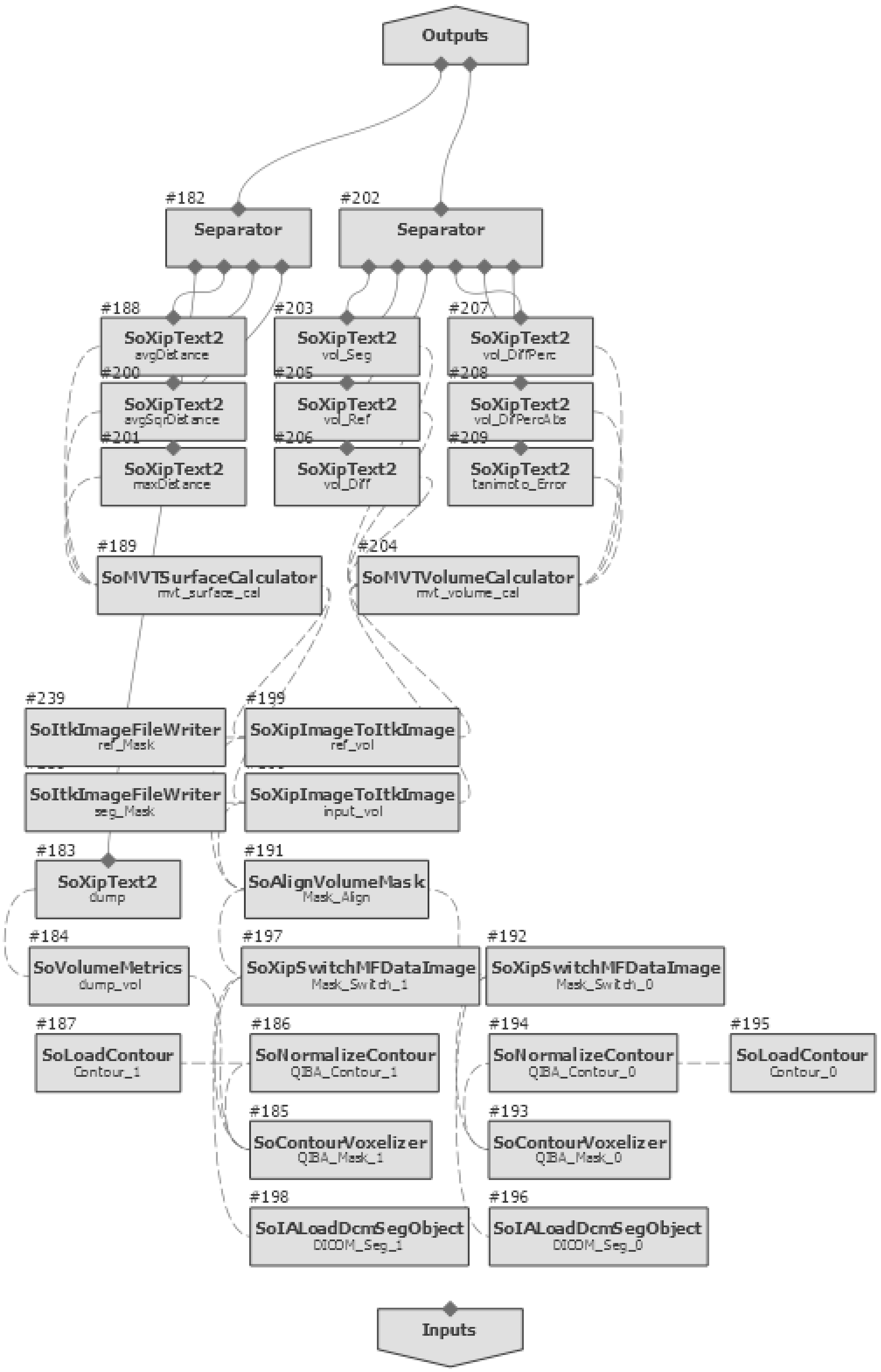


Figure 9 Measurements calculation scene graph

The drilling down function is to compare two annotation results visually via an XIP™ scene graph. The scene graph is shown in Figure 10.

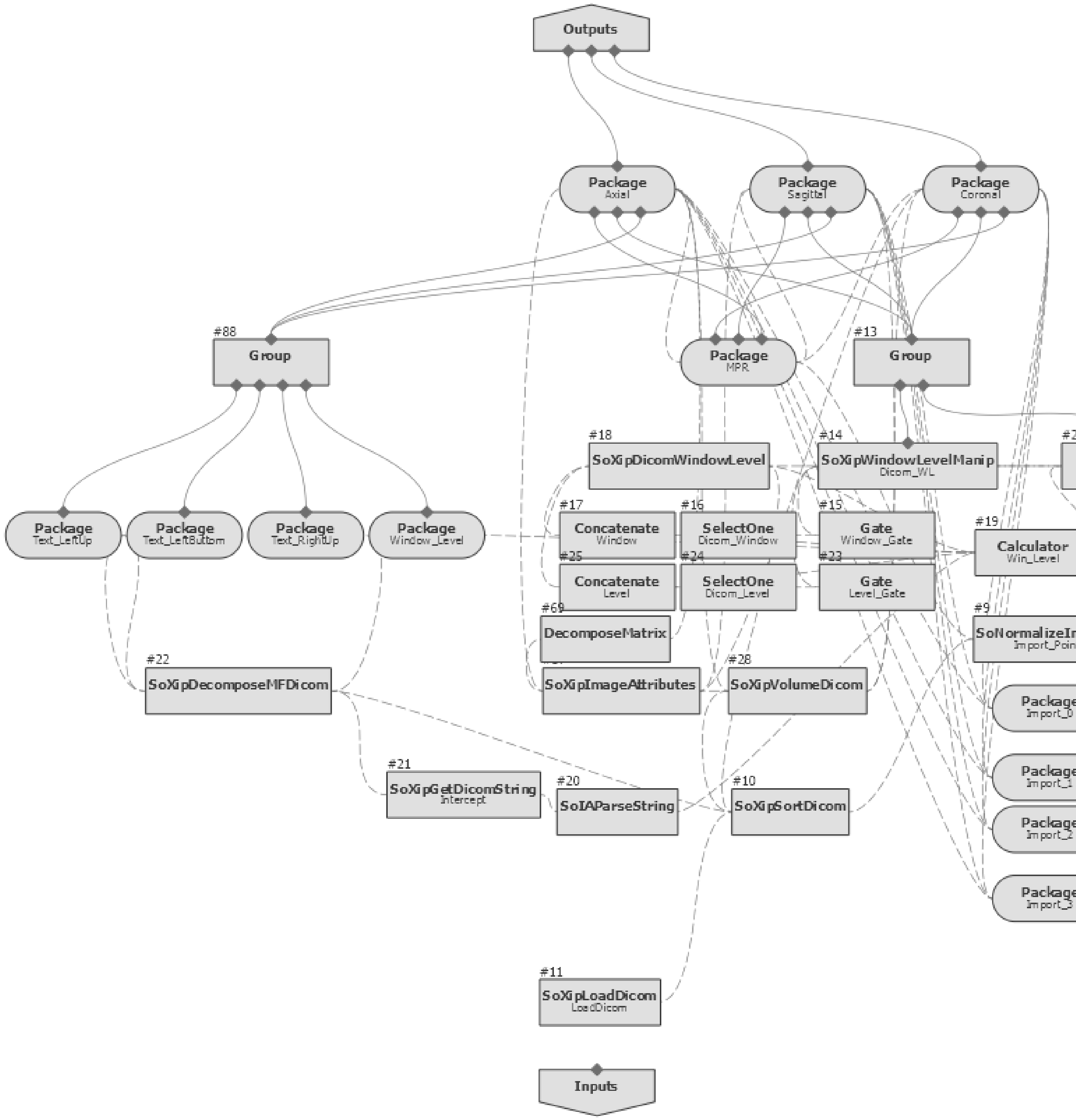


Figure 10 Drilling down scene graph

The MVT has the capability to process four kinds of AIMs, including:

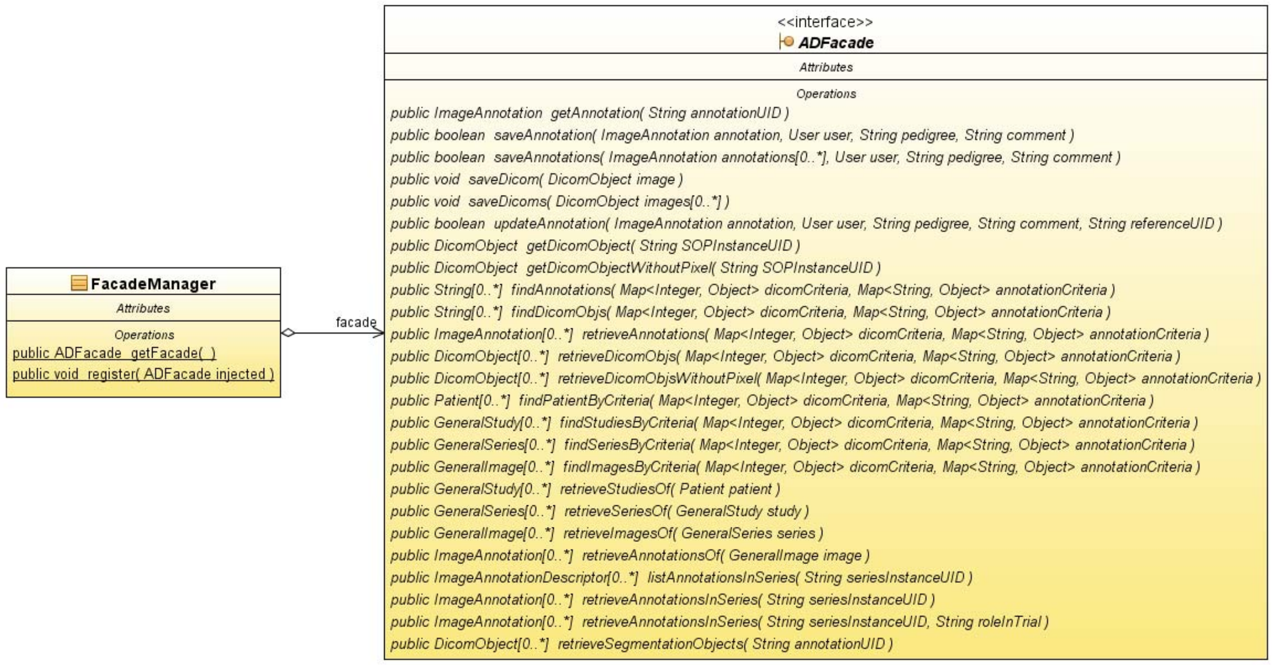
1. RECIST AIM
2. WHO AIM
3. Volume AIM with DICOM segmentation object
4. Volume AIM with geometric contour information

3.5. *AD Core design*

The Assessment Database (AD) provides integrated storage and retrieval of DICOM images and AIM annotations. The annotations and other metadata describing the images is stored and indexed in a relational database for fast access. The AD Core consists of the AD API and the AD implementation. Other components should only use the AD API at compile time while the particular AD implementation created in this project is provided at run time. The AD implementation is using a Hibernate-based mapping of Java objects to a relational database.

3.5.1. AD API

The class ADFacade shown in Figure 21 bundles all functions for storage and retrieval to be used by other components. Annotation objects and DICOM images can be saved to the database and loaded from the database. The find methods accept criteria for AIM as a map from an XPath expression of a value and DICOM criteria as a map from a DICOM header identifier to a value.



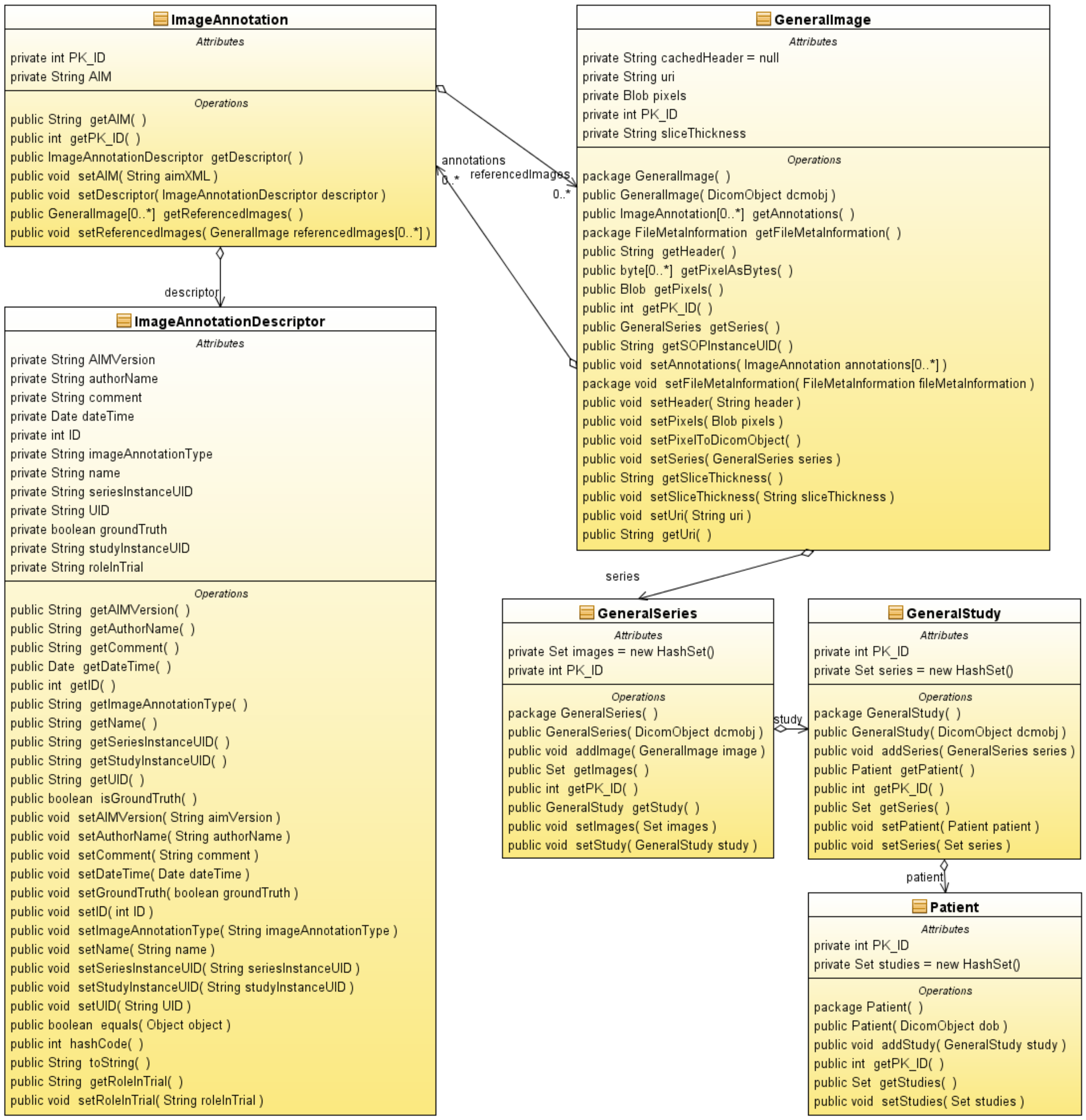
```
public ADFacade  getFacade( )
public void  register( ADFacade injected )
```

```
public ImageAnnotation  getAnnotation( String annotationUID )
public boolean  saveAnnotation( ImageAnnotation annotation, User user, String pedigree, String comment )
public boolean  saveAnnotations( ImageAnnotation annotations[0..*], User user, String pedigree, String comment )
public void  saveDicom( DicomObject image )
public void  saveDicoms( DicomObject images[0..*] )
public boolean  updateAnnotation( ImageAnnotation annotation, User user, String pedigree, String comment, String referenceUID )
public DicomObject  getDicomObject( String SOPInstanceUID )
public DicomObject  getDicomObjectWithoutPixel( String SOPInstanceUID )
public String[0..*]  findAnnotations( Map<Integer, Object> dicomCriteria, Map<String, Object> annotationCriteria )
public String[0..*]  findDicomObjs( Map<Integer, Object> dicomCriteria, Map<String, Object> annotationCriteria )
public ImageAnnotation[0..*]  retrieveAnnotations( Map<Integer, Object> dicomCriteria, Map<String, Object> annotationCriteria )
public DicomObject[0..*]  retrieveDicomObjs( Map<Integer, Object> dicomCriteria, Map<String, Object> annotationCriteria )
public DicomObject[0..*]  retrieveDicomObjsWithoutPixel( Map<Integer, Object> dicomCriteria, Map<String, Object> annotationCriteria )
public Patient[0..*]  findPatientByCriteria( Map<Integer, Object> dicomCriteria, Map<String, Object> annotationCriteria )
public GeneralStudy[0..*]  findStudiesByCriteria( Map<Integer, Object> dicomCriteria, Map<String, Object> annotationCriteria )
public GeneralSeries[0..*]  findSeriesByCriteria( Map<Integer, Object> dicomCriteria, Map<String, Object> annotationCriteria )
public GeneralImage[0..*]  findImagesByCriteria( Map<Integer, Object> dicomCriteria, Map<String, Object> annotationCriteria )
public GeneralStudy[0..*]  retrieveStudiesOf( Patient patient )
public GeneralSeries[0..*]  retrieveSeriesOf( GeneralStudy study )
public GeneralImage[0..*]  retrieveImagesOf( GeneralSeries series )
public ImageAnnotation[0..*]  retrieveAnnotationsOf( GeneralImage image )
public ImageAnnotationDescriptor[0..*]  listAnnotationsInSeries( String seriesInstanceUID )
public ImageAnnotation[0..*]  retrieveAnnotationsInSeries( String seriesInstanceUID )
public ImageAnnotation[0..*]  retrieveAnnotationsInSeries( String seriesInstanceUID, String roleInTrial )
public DicomObject[0..*]  retrieveSegmentationObjects( String annotationUID )
```

Figure 21 AD API

The most important remaining classes of the AD API are shown in Figure 13. The class ImageAnnotation models an AIM object in XML representation. The ImageAnnotationDescriptor is a lightweight class holding the meta-information of an image annotation but not the full annotation itself. The class GeneralImage represents a DICOM image. An annotation can reference one or more images and an image can be associated with one or more annotations. The classes GeneralSeries, GeneralStudy, and Patient represent a grouping of images induced by the corresponding DICOM header tags.

Figure 13 AD API classes



```
ImageAnnotation
Attributes
private int PK_ID
private String AIM
Operations
public String  getAIM( )
public int  getPK_ID( )
public ImageAnnotationDescriptor  getDescriptor( )
public void  setAIM( String aimXML )
public void  setDescriptor( ImageAnnotationDescriptor descriptor )
public GeneralImage[0..*]  getReferencedImages( )
public void  setReferencedImages( GeneralImage referencedImages[0..*] )
```

```
ImageAnnotationDescriptor
Attributes
private String AIMVersion
private String authorName
private String comment
private Date dateTime
private int ID
private String imageAnnotationType
private String name
private String seriesInstanceUID
private String UID
private boolean groundTruth
private String studyInstanceUID
private String roleInTrial
Operations
public String  getAIMVersion( )
public String  getAuthorName( )
public String  getComment( )
public Date  getDateTime( )
public int  getID( )
public String  getImageAnnotationType( )
public String  getName( )
public String  getSeriesInstanceUID( )
public String  getStudyInstanceUID( )
public String  getUID( )
public boolean  isGroundTruth( )
public void  setAIMVersion( String aimVersion )
public void  setAuthorName( String authorName )
public void  setComment( String comment )
public void  setDateTime( Date dateTime )
public void  setGroundTruth( boolean groundTruth )
public void  setID( int ID )
public void  setImageAnnotationType( String imageAnnotationType )
public void  setName( String name )
public void  setSeriesInstanceUID( String seriesInstanceUID )
public void  setStudyInstanceUID( String studyInstanceUID )
public void  setUID( String UID )
public boolean  equals( Object object )
public int  hashCode( )
public String  toString( )
public String  getRoleInTrial( )
public void  setRoleInTrial( String roleInTrial )
```

```
GeneralImage
Attributes
private String cachedHeader = null
private String uri
private Blob pixels
private int PK_ID
private String sliceThickness
Operations
package GeneralImage( )
public GeneralImage( DicomObject dcmobj )
public ImageAnnotation[0..*]  getAnnotations( )
package FileMetaInformation  getFileMetaInformation( )
public String  getHeader( )
public byte[0..*]  getPixelAsBytes( )
public Blob  getPixels( )
public int  getPK_ID( )
public GeneralSeries  getSeries( )
public String  getSOPInstanceUID( )
public void  setAnnotations( ImageAnnotation annotations[0..*] )
package void  setFileMetaInformation( FileMetaInformation fileMetaInformation )
public void  setHeader( String header )
public void  setPixels( Blob pixels )
public void  setPixelToDicomObject( )
public void  setSeries( GeneralSeries series )
public String  getSliceThickness( )
public void  setSliceThickness( String sliceThickness )
public void  setUri( String uri )
public String  getUri( )
```

```
GeneralSeries
Attributes
private Set images = new HashSet()
private int PK_ID
Operations
package GeneralSeries( )
public GeneralSeries( DicomObject dcmobj )
public void  addImage( GeneralImage image )
public Set  getImages( )
public int  getPK_ID( )
public GeneralStudy  getStudy( )
public void  setImages( Set images )
public void  setStudy( GeneralStudy study )
```

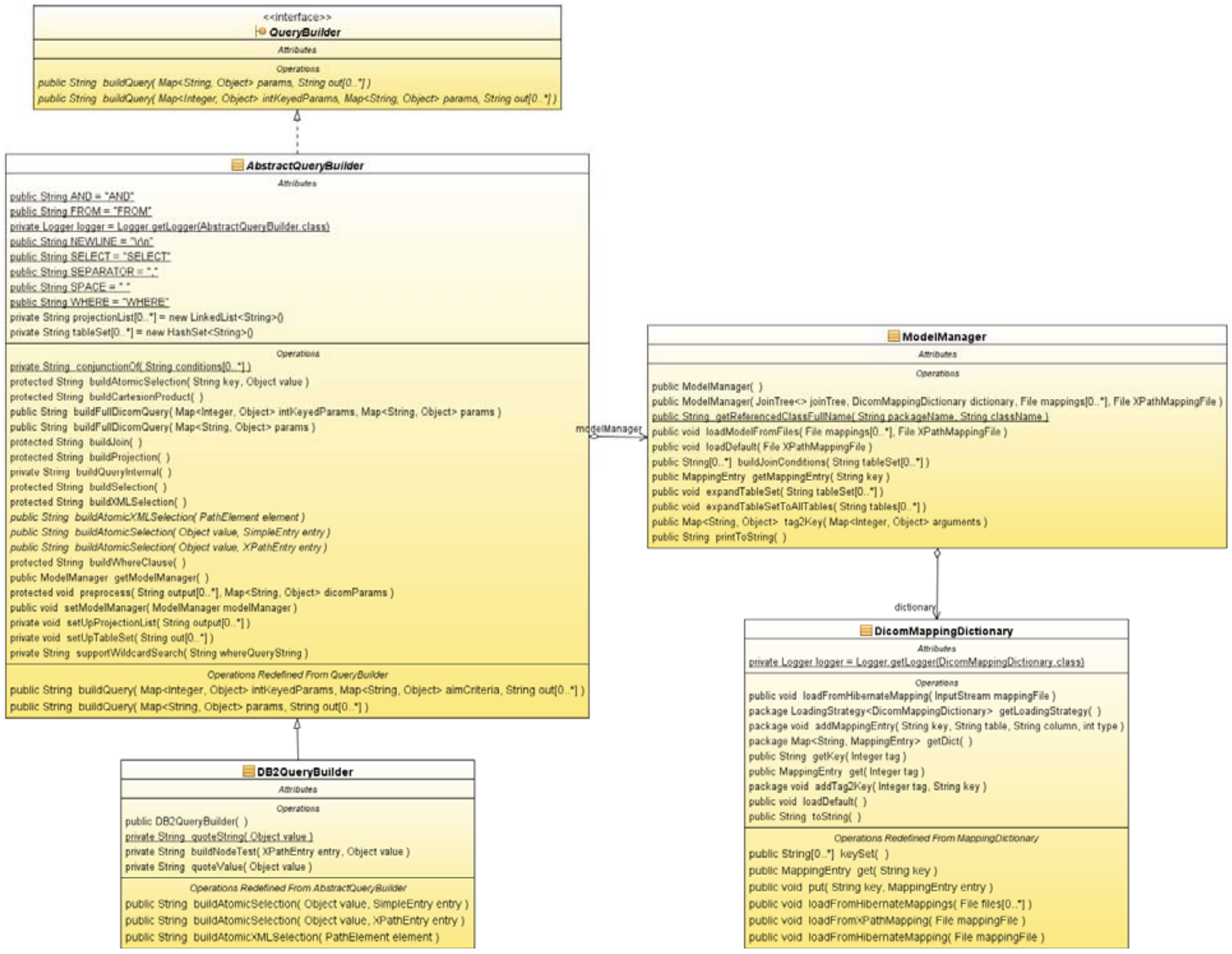
```
GeneralStudy
Attributes
private int PK_ID
private Set series = new HashSet()
Operations
package GeneralStudy( )
public GeneralStudy( DicomObject dcmobj )
public void  addSeries( GeneralSeries series )
public Patient  getPatient( )
public int  getPK_ID( )
public Set  getSeries( )
public void  setPatient( Patient patient )
public void  setSeries( Set series )
```

```
Patient
Attributes
private int PK_ID
private Set studies = new HashSet()
Operations
package Patient( )
public Patient( DicomObject dob )
public void  addStudy( GeneralStudy study )
public int  getPK_ID( )
public Set  getStudies( )
public void  setStudies( Set studies )
```

3.5.2. AD Implementation

The internal classes of the AD implementation are shown in Figure 14. The formulation of database queries against the combined AIM and DICOM database is modeled by the QueryBuilder. Based on the criteria passed through the ADFacade, conjunctive queries are formulated against the database automatically joining the tables holding annotations and images using the references inside the AIM XML content. The class AbstractQueryBuilder contains all methods that are independent of the particular database used. The DB2QueryBuilder contains the methods that provide the AD query functionality based in the IBM DB2 database that supports efficient querying of XML content using the XPath query language.

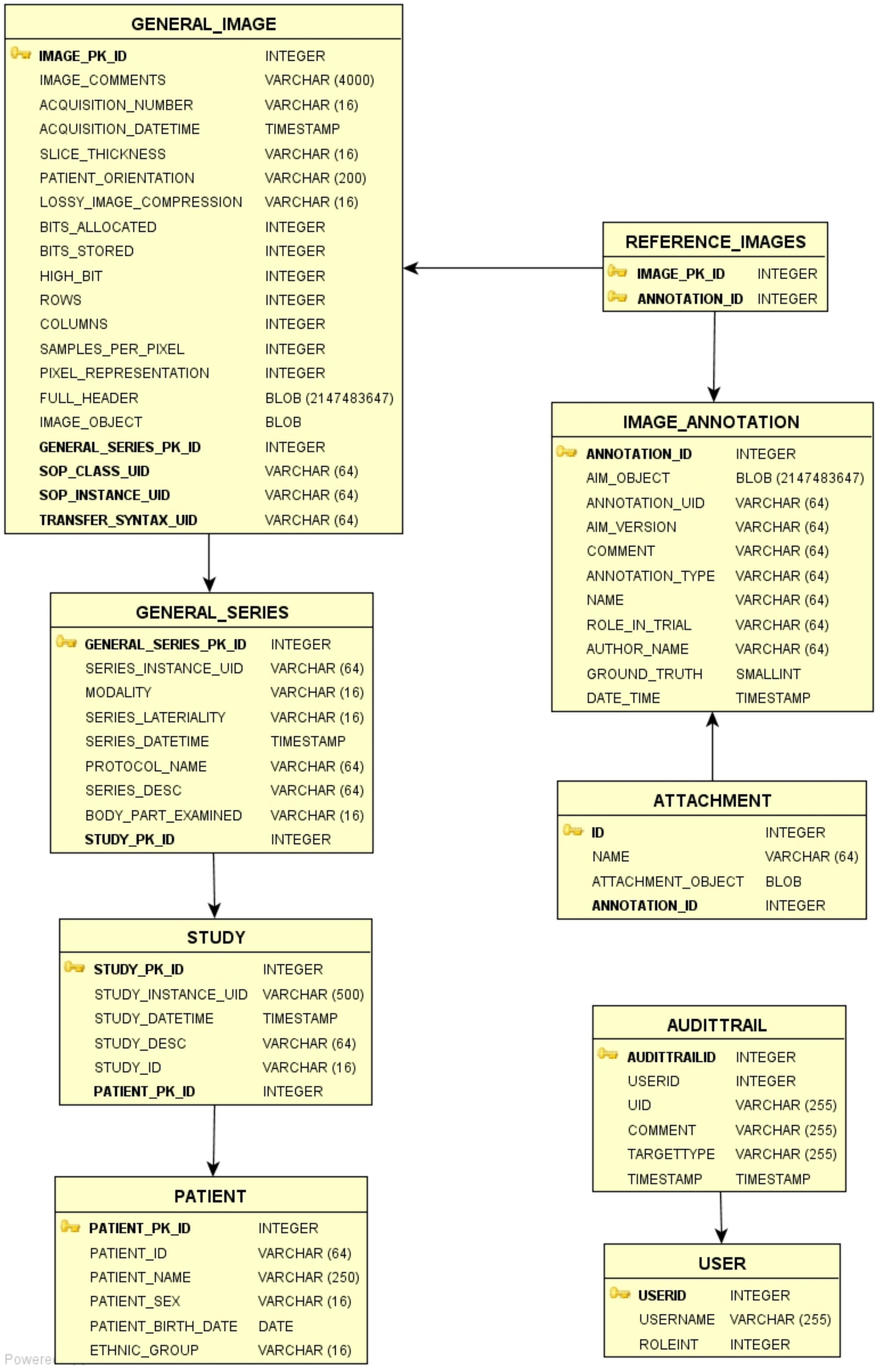
Figure 14 AD Implementation


<<interface>>
QueryBuilder
Attributes
Operations
public String buildQuery(Map<String, Object> params, String out[0..*])
public String buildQuery(Map<Integer, Object> intKeyedParams, Map<String, Object> params, String out[0..*])
AbstractQueryBuilder
Attributes
public String AND = "AND"
public String FROM = "FROM"
private Logger logger = Logger.getLogger(AbstractQueryBuilder.class)
public String NEWLINE = "\r\n"
public String SELECT = "SELECT"
public String SEPARATOR = ","
public String SPACE = " "
public String WHERE = "WHERE"
private String projectionList[0..*] = new LinkedList<String>()
private String tableSet[0..*] = new HashSet<String>()
Operations
private String conjunctionOf(String conditions[0..*])
protected String buildAtomicSelection(String key, Object value)
protected String buildCartesionProduct( )
public String buildFullDicomQuery(Map<Integer, Object> intKeyedParams, Map<String, Object> params)
public String buildFullDicomQuery(Map<String, Object> params)
protected String buildJoin( )
protected String buildProjection( )
private String buildQueryInternal( )
protected String buildSelection( )
protected String buildXMLSelection( )
public String buildAtomicXMLSelection(PathElement element)
public String buildAtomicSelection(Object value, SimpleEntry entry)
public String buildAtomicSelection(Object value, XPathEntry entry)
protected String buildWhereClause( )
public ModelManager getModelManager( )
protected void preprocess(String output[0..*], Map<String, Object> dicomParams)
public void setModelManager(ModelManager modelManager)
private void setUpProjectionList(String output[0..*])
private void setUpTableSet(String out[0..*])
private String supportWildcardSearch(String whereQueryString)
Operations Redefined From QueryBuilder
public String buildQuery(Map<Integer, Object> intKeyedParams, Map<String, Object> aimCriteria, String out[0..*])
public String buildQuery(Map<String, Object> params, String out[0..*])
modelManager
ModelManager
Attributes
Operations
public ModelManager( )
public ModelManager(JoinTree<> joinTree, DicomMappingDictionary dictionary, File mappings[0..*], File XPathMappingFile)
public String getReferencedClassFullName(String packageName, String className)
public void loadModelFromFiles(File mappings[0..*], File XPathMappingFile)
public void loadDefault(File XPathMappingFile)
public String[0..*] buildJoinConditions(String tableSet[0..*])
public MappingEntry getMappingEntry(String key)
public void expandTableSet(String tableSet[0..*])
public void expandTableSetToAllTables(String tables[0..*])
public Map<String, Object> tag2Key(Map<Integer, Object> arguments)
public String printToString( )
dictionary
DicomMappingDictionary
Attributes
private Logger logger = Logger.getLogger(DicomMappingDictionary.class)
Operations
public void loadFromHibernateMapping(InputStream mappingFile)
package LoadingStrategy<DicomMappingDictionary> getLoadingStrategy( )
package void addMappingEntry(String key, String table, String column, int type)
package Map<String, MappingEntry> getDict( )
public String getKey(Integer tag)
public MappingEntry get(Integer tag)
package void addTag2Key(Integer tag, String key)
public void loadDefault( )
public String toString( )
Operations Redefined From MappingDictionary
public String[0..*] keySet( )
public MappingEntry get(String key)
public void put(String key, MappingEntry entry)
public void loadFromHibernateMappings(File files[0..*])
public void loadFromXPathMapping(File mappingFile)
public void loadFromHibernateMapping(File mappingFile)
DB2QueryBuilder
Attributes
Operations
public DB2QueryBuilder( )
private String quoteString(Object value)
private String buildNodeTest(XPathEntry entry, Object value)
private String quoteValue(Object value)
Operations Redefined From AbstractQueryBuilder
public String buildAtomicSelection(Object value, SimpleEntry entry)
public String buildAtomicSelection(Object value, XPathEntry entry)
public String buildAtomicXMLSelection(PathElement element)

3.6. AD Installer design

The AD installer package is a small standalone command line tool that can create and update an SQL database with the AD database schema shown in Figure 15.

Figure 15 AD database schema



GENERAL_IMAGE	
IMAGE_PK_ID	INTEGER
IMAGE_COMMENTS	VARCHAR (4000)
ACQUISITION_NUMBER	VARCHAR (16)
ACQUISITION_DATETIME	TIMESTAMP
SLICE_THICKNESS	VARCHAR (16)
PATIENT_ORIENTATION	VARCHAR (200)
LOSSY_IMAGE_COMPRESSION	VARCHAR (16)
BITS_ALLOCATED	INTEGER
BITS_STORED	INTEGER
HIGH_BIT	INTEGER
ROWS	INTEGER
COLUMNS	INTEGER
SAMPLES_PER_PIXEL	INTEGER
PIXEL_REPRESENTATION	INTEGER
FULL_HEADER	BLOB (2147483647)
IMAGE_OBJECT	BLOB
GENERAL_SERIES_PK_ID	INTEGER
SOP_CLASS_UID	VARCHAR (64)
SOP_INSTANCE_UID	VARCHAR (64)
TRANSFER_SYNTAX_UID	VARCHAR (64)

REFERENCE_IMAGES	
IMAGE_PK_ID	INTEGER
ANNOTATION_ID	INTEGER

IMAGE_ANNOTATION	
ANNOTATION_ID	INTEGER
AIM_OBJECT	BLOB (2147483647)
ANNOTATION_UID	VARCHAR (64)
AIM_VERSION	VARCHAR (64)
COMMENT	VARCHAR (64)
ANNOTATION_TYPE	VARCHAR (64)
NAME	VARCHAR (64)
ROLE_IN_TRIAL	VARCHAR (64)
AUTHOR_NAME	VARCHAR (64)
GROUND_TRUTH	SMALLINT
DATE_TIME	TIMESTAMP

GENERAL_SERIES	
GENERAL_SERIES_PK_ID	INTEGER
SERIES_INSTANCE_UID	VARCHAR (64)
MODALITY	VARCHAR (16)
SERIES_LATERIALITY	VARCHAR (16)
SERIES_DATETIME	TIMESTAMP
PROTOCOL_NAME	VARCHAR (64)
SERIES_DESC	VARCHAR (64)
BODY_PART_EXAMINED	VARCHAR (16)
STUDY_PK_ID	INTEGER

ATTACHMENT	
ID	INTEGER
NAME	VARCHAR (64)
ATTACHMENT_OBJECT	BLOB
ANNOTATION_ID	INTEGER

STUDY	
STUDY_PK_ID	INTEGER
STUDY_INSTANCE_UID	VARCHAR (500)
STUDY_DATETIME	TIMESTAMP
STUDY_DESC	VARCHAR (64)
STUDY_ID	VARCHAR (16)
PATIENT_PK_ID	INTEGER

AUDITTRAIL	
AUDITTRAILID	INTEGER
USERID	INTEGER
UID	VARCHAR (255)
COMMENT	VARCHAR (255)
TARGETTYPE	VARCHAR (255)
TIMESTAMP	TIMESTAMP

PATIENT	
PATIENT_PK_ID	INTEGER
PATIENT_ID	VARCHAR (64)
PATIENT_NAME	VARCHAR (250)
PATIENT_SEX	VARCHAR (16)
PATIENT_BIRTH_DATE	DATE
ETHNIC_GROUP	VARCHAR (16)

USER	
USERID	INTEGER
USERNAME	VARCHAR (255)
ROLEINT	INTEGER

3.7. XIP™ design

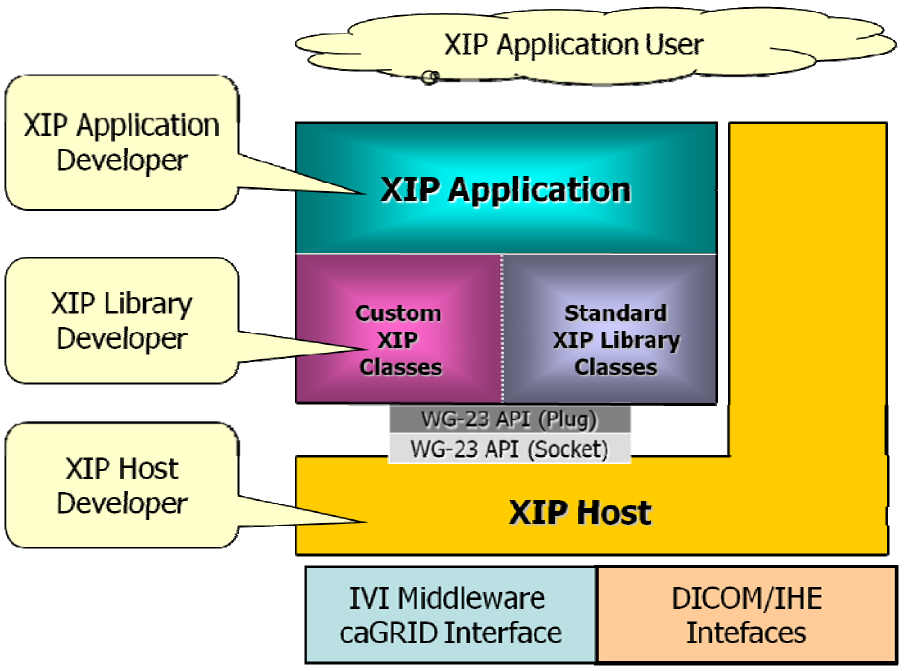
XIP™ is a development/deployment environment, commissioned by NCI's caBIG® program, that facilitates the rapid creation of portable, task specific applications. XIP follows the Application Hosting paradigm described in DICOM Supplement 118 (created by DICOM WG-23), where infrastructure common to all applications, such as network access, database access, and security, are separated into a Hosting System. This Hosting System then launches and manages a set of Hosted Applications, assigning them tasks, supplying them with data, collecting the results that they create, and storing those results into a database or network service. Since the Hosting System manages the infrastructure, the Hosted Applications need only worry about the business logic of the application. Please refer to DICOM Supplement 118 for a description of the DICOM Application Hosting Interfaces.

The XIP run time environment (e.g. the AVT workstation) consists of three principle software components (see figure 1):

1. The XIP Host™ – the XIP Host™ software implements the Hosting System side of the DICOM Application Hosting interfaces, and supplies the common infrastructure utilized by all applications. For AVT, the XIP Host™ is extended to allow access to data within the AD.

2. The XIP Libraries™ – the library modules that provide image processing and visualization functions that an application developer may use to create new XIP™ Applications. Most of the XIP Libraries are provided as extensions to the Open Inventor™ libraries, and are utilized via the building of scene graphs and pipelines. Certain AVT functions are built as extensions to the XIP Libraries™.

3. Multiple XIP™ Applications – the software that supplies the business logic for performing particular tasks, generally through the use of the XIP Libraries™. XIP Applications in particular should react to actions triggered by and data provided by the XIP Host™, as well as return any results and report events back to the XIP Host through the DICOM Application Hosting interface. The XIP™ software distribution provides Java classes for building XIP™ Applications, including an extension to the Java . These base classes manage the Hosted Application side of the DICOM Application Hosting interfaces. The AVT IA, AE, and MVT applications are built upon these base classes, and utilize the XIP Libraries™ with AVT extensions for processing and visualization.



3.7.1. XIP Host™

The XIP Host is layered on top of the IVI Middleware for caGrid connectivity and security, coupled with the PixelMed DICOM Toolkit for interactions with other DICOM Application Entities on the network.

GUI interactions within the XIP Host™ occur through a set of top level GUI classes. Long running operations (e.g. query or retrieve of data) are spun off into separate threads as Java runnable classes, both to avoid blocking the GUI and to improve performance. Manager classes keep track of these threaded operations, triggering appropriate GUI responses as operations complete.

The GUI for the XIP Host is built using Java Swing and provides access to controls that allow a user to:

- search for data within AD, on caGrid Image Data Services, on DICOM Storage Services, or on the local file system,
- retrieve the data found and provide it as input to XIP applications, including the IA, AE and MVT, that the user launches,
- switch between running XIP applications,
- configure various aspects of the XIP host,

The XIP Host internally includes logic to track the running applications, and accept results data from them through the DICOM Application Hosting interfaces. It uses the JAX-WS system that is part of the Sun Java 6.0 distribution to implement those interfaces. The DICOM Application

Hosting interfaces are described in Supplement 118 of the DICOM Standard. The WSDL and XML schema definitions, as well as UML model diagrams of these interfaces are included in the XIP source code distribution.

3.7.2. XIP Libraries™

The main portions of the XIP Application Libraries are extensions built on top of the Open Inventor™ framework. They are written in c++ for compatibility with the base Open Inventor™ libraries, and for optimal performance. The design and use of the Open Inventor framework is described in the book "The Inventor Mentor" (**ISBN-13:** 978-0201624953). The book "The Inventor Toolmaker" (**ISBN-13:** 978-0201624939) describes how to extend the base Open Inventor libraries. Those two books are the primary documentation on the how the Open Inventor extensions within the XIP Libraries™ are built.

XIP™ Release 0.2.2, utilized by AVT2EXT, includes the following Open Inventor™ extension libraries:

- XipIvCore: An Inventor extension library for basic data types needed in medical imaging.
- XipIvCoreGL: An extension library for enhanced OpenGL features that are not supported in the original Open Inventor™ library.
- XipIvDicom: A library for DICOM object parsing based on the Offis DCMTK toolkit.
- XipIvITK: A library of scene objects drawn from the Insight Segmentation and Registration Toolkit (ITK).
- XipIvOverlay: A library providing a set of overlay (markup) objects for annotating images and performing measurements.
- XipIvRenderer: A modular, configurable volume renderer for three-dimensional image data.
- XipIvVTK: A library of scene objects drawn from the Visualization Toolkit (VTK).

3.7.3. XIP™ Application Support Java Classes

In addition to the Open Inventor™ extensions, XIP™ includes two Java classes, with associated support routines and classes, to assist in building XIP™ Applications:

- ivCanvas: A JNI-based class, derived from the Java AWT canvas class, bridging to the Open Inventor™ libraries which provides:
    - a drawing area for use by Open Inventor™ scene graphs,
    - a means to load Open Inventor™ extension libraries,
    - a means to load Open Inventor™ scene graphs,
    - a means to interact with fields in modules that are part of the scene graph.

- WG23Application: A class that provides the DICOM Application Hosting interface from the Hosted Application to the Hosting System, which application developers can use as a base class to derive their application code.

3.8. Safety

N/A. AVT has no safety-critical requirements.

3.9. Dynamic behavior

N/A

3.10. Interface Topics

3.10.1. Image Reader UI State Transition design

The UI state transition diagram is shown in Figure 15.

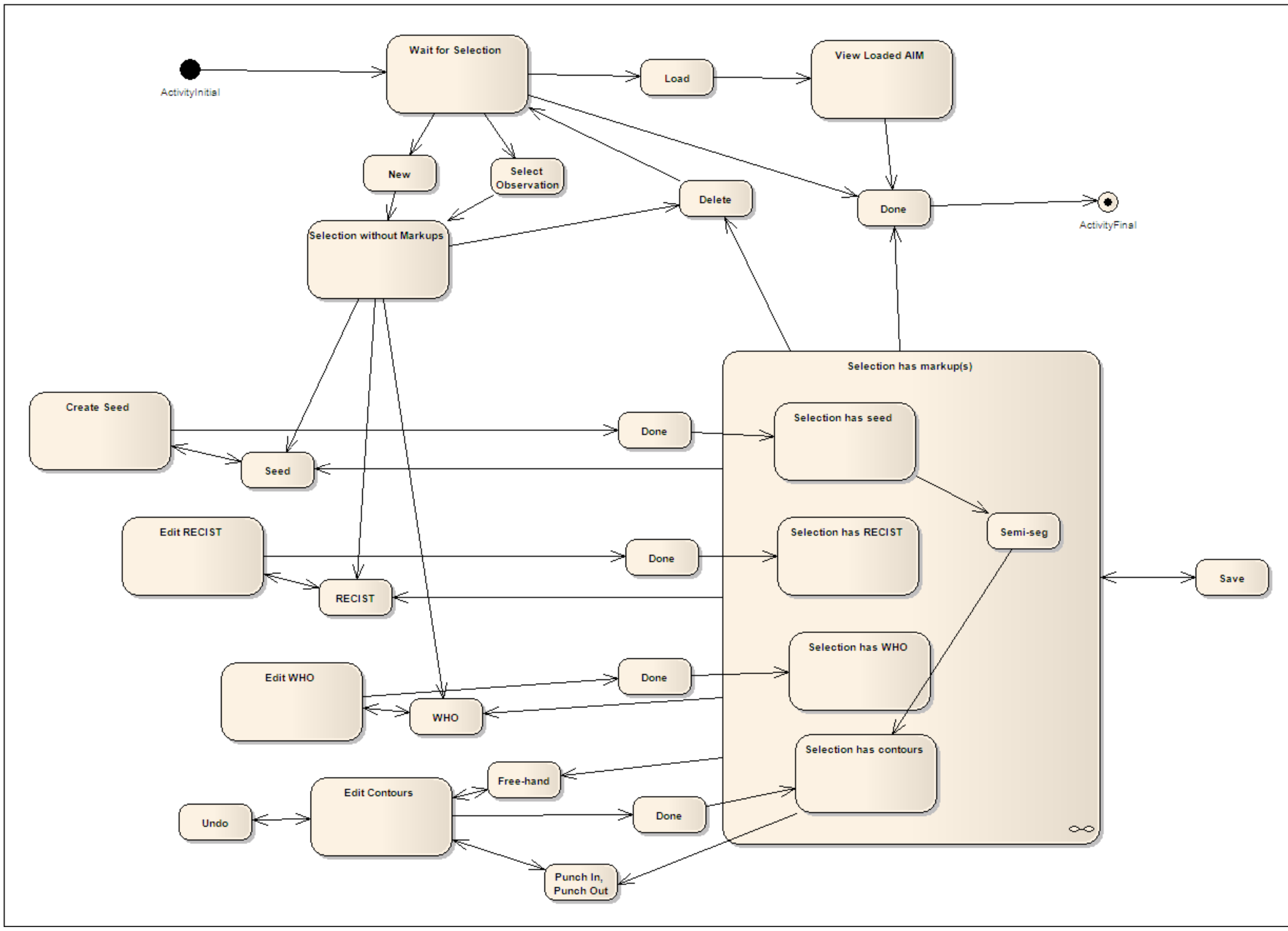


Figure 15 IA UI state transition

3.10.2. Hardware interfaces

N/A

3.10.3. Operating system interfaces

N/A

3.10.4. External software interfaces

N/A

3.10.5. R Library Interfaces

The Rserve, a TCP/IP server which allows other programs to use facilities of R (see www.r-project.org) without the need to link against R library, runs as a separated process in MVT to interact with R package to perform the statistical analysis.

The XIP™ scene graph used as the Rserve client-side implementation is shown in Figure 16.

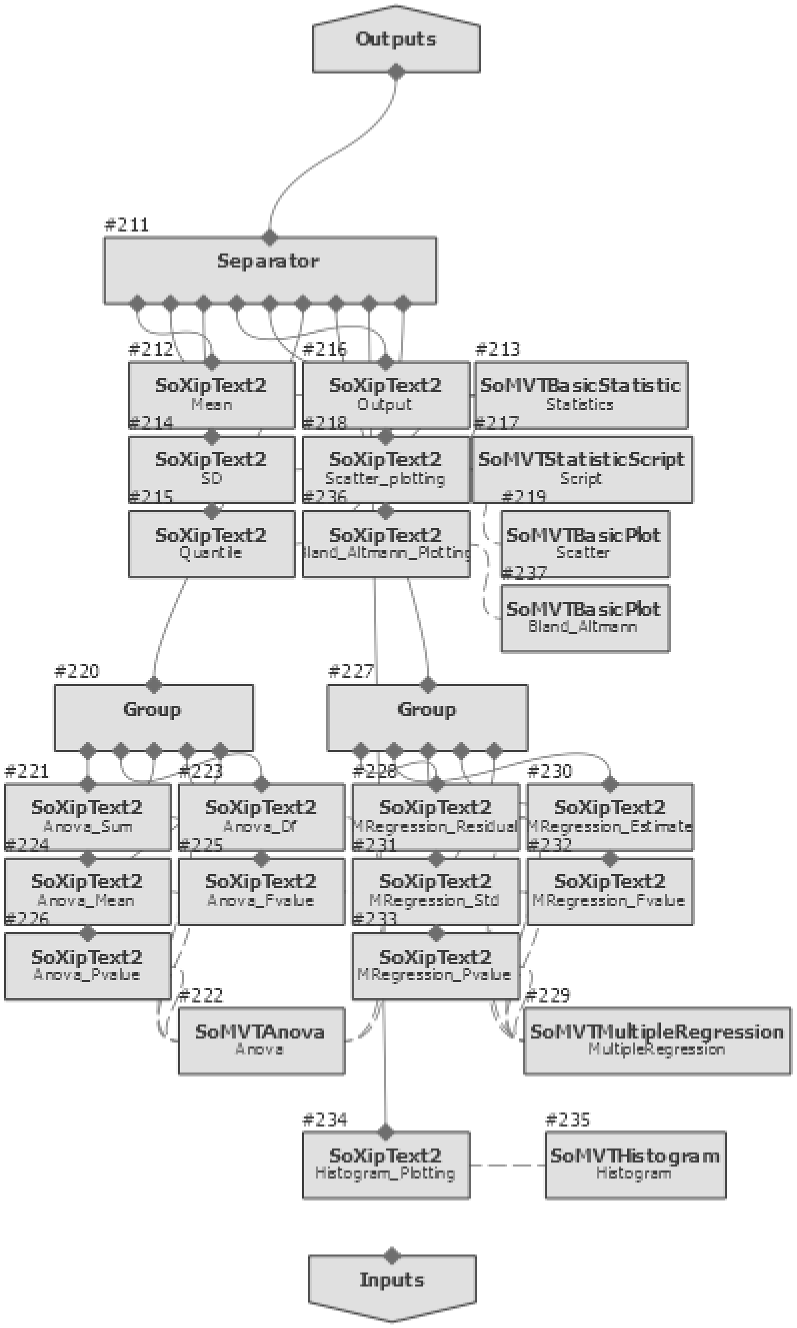


Figure 16 R statistical analysis scene graph

3.10.6. XIP Libraries™ Interfaces

Revision 0.2.2 of the open source XIP Libraries™ is used to develop the AVT application extensions and to design the scene graph.

3.11. Internal software interfaces

N/A

3.12. Data dictionary

N/A

4. Data design topics

4.1. AIM design for CDRH case study

In order to meet the special needs in AVT2EXT, several coding schema extensions are made to the AIM schema defined in "AIM_TCGA09302009_XML.xsd", i.e.

. codingSchemaVersion = "v0rv1"
. coding SchemaDesignator = "AVT2"

4.1.1. Seeds

The Seeds AIM is defined to store the stroke location of the seed points, which is used as one of the input parameters for the segmentation algorithm in IA and AE. The AIM codeValue and codeMeaning are defined as follows,

. codeValue = "AVT001"
. codeMeaning = "Tumor Seed Annotation"

4.1.2. RECIST

The RECIST AIM is defined to store the RECIST measurement of a tumor. The AIM codeValue and codeMeaning are defined as follows,

. codeValue = "AVT002"
. codeMeaning = "Tumor RECIST Annotation"

4.1.3. WHO

The RECIST AIM is defined to store the WHO measurement of a tumor. The AIM codeValue and codeMeaning are defined as follows,

. codeValue = "AVT003"
. codeMeaning = "Tumor WHO Annotation"

4.1.4. volume

There are three volume AIMs defined to store the volume ROI of a tumor. The AIM codeValue and codeMeaning are defined as follows,

i. Volume annotation generated in IA
 . codeValue = "AVT004"
 . codeMeaning = "Tumor volume annotation"
ii. Volume segmentation generated in AE
 . codeValue = "AVT005"
 . codeMeaning = "Tumor volume segmentation"
iii. Volume contour converted from CDRH annotation
 . codeValue = "AVT006"
 . codeMeaning = "Tumor volume contours"

4.1.5. work-arounds

Since there isn't suitable attribute available in current schema to record the annotation session information, the work-around is to append the session to the annotator's name in "user.name" attribute as follows.

. user.name = "annotator name"_"annotation session"

4.1.6. variations

The relationship between imported CDRH AIMs and AVT-produced AIMs as follows,

. They share the same RECIST and WHO AIMs;

. The AVT-produced volume AIMs use a referred DICOM segmentation object to represent the tumor volume binary mask;

. The CDRH AIMs use the geometric points to represent the contour of the tumor.

4.2. AIM design for TCGA case study

See AIM Schema file – "AIM_TCGA09302009_XML.xsd".

4.3. Database design

See Figure "Database Schema"

5. API Documentation

5.1. Application APIs

The API documentation for the IA, AE, and MVT is available in the subfolders IA/doc, AE/doc, and MVT/doc, of the AVT installation.

5.2. AD APIs

The API documentation for the AP API is available in the subfolder ad-api/docs of the AVCT installation.

5.3. XIPHost_AVT

The API documentation for the XIP Host™, including the extensions made to the XIP Host™ for AVT, is embedded in the XIP Host™ source code, and may be generated using the Javadoc system.

6. General quality features

6.1. MISC_operational_pilot

This quality attribute, a type of reliability, is addressed by the incremental, iterative development process, which emphasizes testing early and often to uncover gaps in functionality as well as bugs that interfere with usability.

6.2. MVT_CDRH_case_capacity, MVT_CDRH_annotation_capacity

MVT is designed for a configurable maximum number of cases and number of annotations that are much larger than required for CDRH. We then tested MVT on the CDRH data to confirm that it had sufficient capacity.

6.3. Availability

No availability requirements

6.4. User friendliness

Addressed by testing early and often and by arranging with early adopters to provide user friendliness feedback.

6.5. Maintainability

No maintainability requirements

6.6. Transferability, portability

Addressed by using platforms and libraries that are known for transferability and portability.

6.7. Testability

No testability requirements.

7. Unsettled points / open issues

None.