Coverage Framework Design

A GeoTools Implemetation of ISO 19123

Bryce Nordgren
USDA Forest Service

Legal

Table of Contents

1 Administrative Details

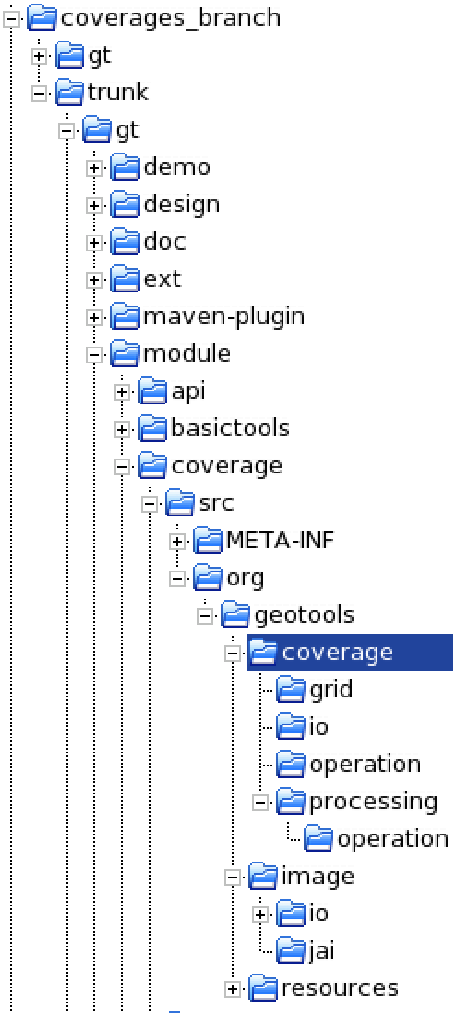
1.1 Transition from OGC to 19123

The implementation of ISO 19123 classes in GeoTools poses a logistical problem due to the fact that there is an existing coverage implementation already in place. This existing implementation consumes the org.geotools.coverage namespace, as shown in Figure 1. The existing coverage implementation tracks the Open Geospatial Consortium's (OGC's) grid coverage specification, given in OGC document 01-004. This specification is being retired in favor of an ISO 19123 based coverage specification.

No determination of the fate of these packages has yet been made. Presumably, they will be deprecated in the first release which contains the ISO 19123 coverage implementation, and removed in the following release. In any case, there will be a period of time during which the two implementations are required to coexist in the same namespace.

Wherever the coverage implementation is rooted, a direct implementation of 19123 will require seven packages when complete, and three for the immediate implementation effort. These packages are (**bold** package names are needed immediately):

1. **Coverage Core**
2. **Quadrilateral Grid**
3. Hexagonal Grid
4. **Discrete Coverages**
5. Thiessen Polygon
6. TIN
7. Segmented Curve



```
coverages_branch
  gt
  trunk
    gt
      demo
      design
      doc
      ext
      maven-plugin
      module
        api
        basictools
        coverage
          src
            META-INF
            org
              geotools
                coverage
                  grid
                  io
                  operation
                  processing
                    operation
                image
                  io
                  jai
                resources
```

Figure 1: Coverage Packages currently in the GeoTools namespace

These packages may be accommodated with the mapping from ISO 19123 packages to java namespace shown in Table 1. This mapping satisfactorily avoids contention with existing classes and provides a namespace for a complete implementation of the specification. Yellow cells in the table represent packages which will be completely or partially implemented as a result of this initial effort.

Package Name	***Namespace***
Coverage Core	org.geotools.coverage.core
Quadrilateral Grid	org.geotools.coverage.qgrid
Hexagonal Grid	org.geotools.coverage.hgrid
Discrete Coverages	org.geotools.coverage.discrete
Thiessen Polygon	org.geotools.coverage.thiessen
TIN	org.geotools.coverage.tin
Segmented Curve	org.geotools.coverage.curve

Table 1: Mapping of ISO 19123 package name to GeoTools namespace

1.2 Scope

The scope of this initial implementation effort is to get some basic coverage functionality integrated into GeoTools. As shown in Table 1, less than half of the coverage packages are scheduled even for partial implementation. The focus of the current effort is to replace the current GeoTools implementation of the OGC GridCoverage specification with an nD implementation based on ISO 19123. The primary (if not *only*) use of the OGC coverage implementation was to load, process and render various raster data formats (e.g., Arc/Info ASCII Grid, GeoTIFF, World File, etc.) Even though a small fraction of ISO 19123 is scheduled for immediate implementation, the overall impact will be to increase the effectiveness of the GeoTools coverage capability.

Coverage Core and Quadrilateral Grid are to be completely implemented. The Discrete Coverages package will be implemented to such an extent that CV_DiscretePointCoverage and CV_DiscreteGridPointCoverage are fully specified and operational.

There are three items not in ISO 19123 which are part of this implementation effort. These are:

1. The proposed extension to ISO 19123 for the separation of the axes of a multidimensional grid into a set of domain axes and a set of range axes. This extension results in the creation of two back-ends for the discrete grid point coverage type, as documented in section 5.

2. An abstract class to facilitate the creation of analytical coverages. This is documented in section 6.6.

3. The abstraction of spatial indices for use with discrete coverages. This is necessary to provide for the efficient spatio-temporal-based storage and lookup of discrete point coverage data. This is documented in section 4.1.

This implementation document has many sections which are “stubbed out” and will remain “stubbed out” even when the current implementation is considered complete. This document is meant to collect together in one place the design and extent of the initial implementation. It is meant to be a design reference for users who want to know what choices were made during the implementation. It is also intended to be modified whenever the implementation is fleshed out more fully by some future implementor (e.g., by the addition of a TIN coverage.)

1.3 Use of Eclipse Modeling Framework

This implementation effort is starting before the feature model is finalized. Conceptually, coverages are specializations of features. As the feature model is being finalized the decision of whether to make Coverages actually extend Features has been put on hold.

Regardless, one of the salient features of any dynamic feature type is a provision for uniform data access. The generated code from the Eclipse Modeling Framework (EMF) provides just such a hook. Since this is desirable without regard to the actual feature model, all coverage development work will proceed by defining an Ecore model, generating the model code, and customizing the model as needed. The Ecore model will be preserved in Subversion as part of the design.

2 Definitions

Term	*Definition*
Adapter pattern	The adapter pattern converts the interface of a class into another interface the clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

3 Coverage Core

4 Discrete Coverages

Discrete coverages are essentially feature collections where all of the contained objects have exactly the same feature type. The responsibility of a discrete coverage is to present the contents of this collection for efficient retrieval of one or more elements with reference to the coverage's spatio-temporal coordinates. All members of a coverage are referenced to the same CRS.

4.1 Spatial Indices

4.2 Discrete Point Coverage

4.3 Discrete Grid Point Coverage

4.4 Discrete Curve Coverage

POSTPONE: Implementation of a discrete curve coverage is deferred to a future date.

4.5 Discrete Surface Coverage

POSTPONE: Implementation of a discrete surface coverage is deferred to a future date.

4.6 Discrete Solid Coverage

POSTPONE: Implementation of a discrete solid coverage is deferred to a future date.

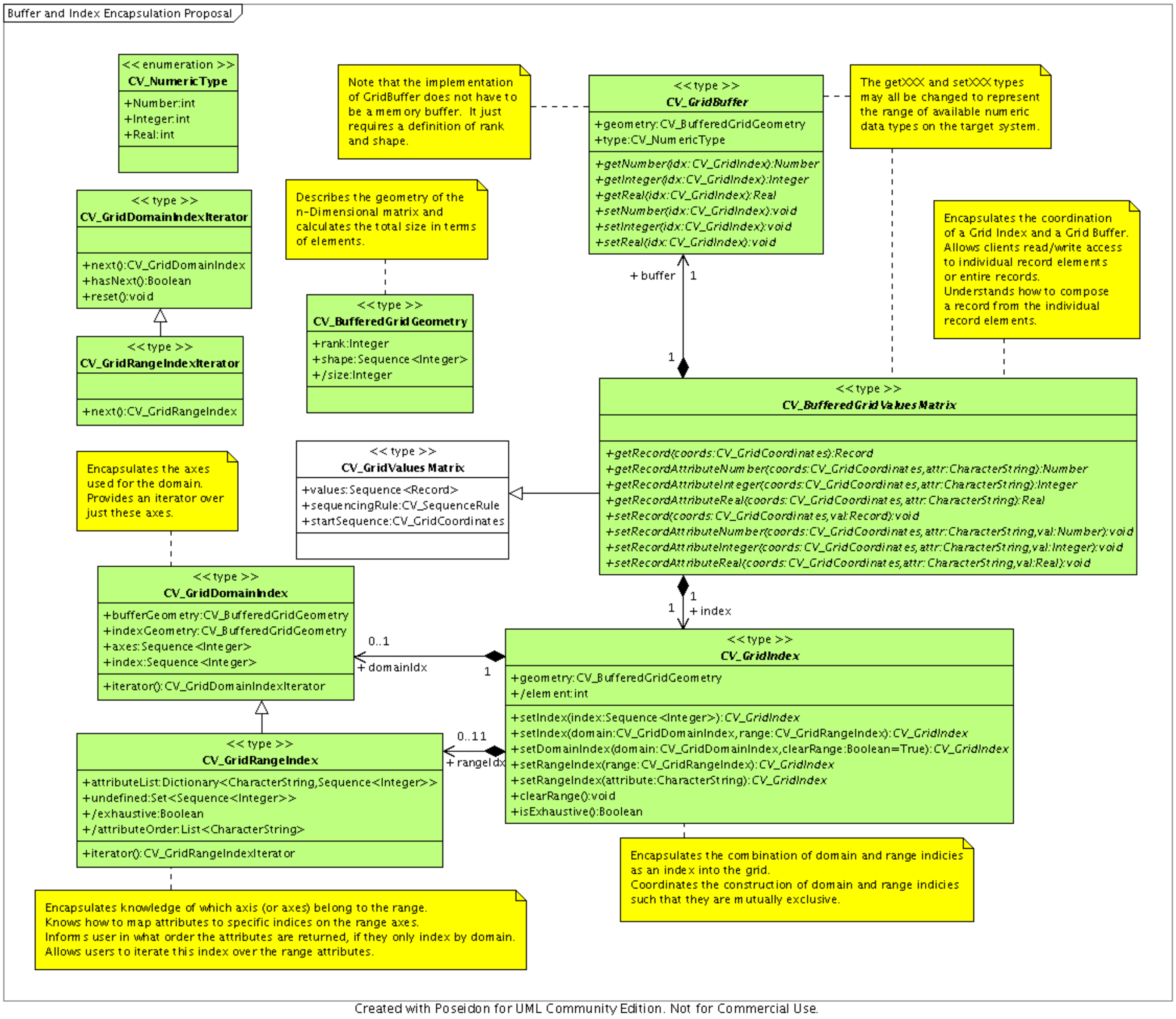
5 Gridded Coverage Extensions

This section deals exclusively with the proposed extension to ISO19123 found in the ISO 19123 Primer. This extension provides a common API for the handling of regularly gridded data with homogeneous data attributes. The form of this extension is illustrated in Figure 2.

Two implementations are outlined. An implementation to handle two dimensional data, backed by J2SE ImageIO is the subject of section 5.2. An implementation to handle arbitrary multidimensional data, backed by the Multiarray2 library, is the subject of section 5.3. Code common to the two implementations is presented in section 5.1.

The overall design presented in Figure 2 facilitates the introduction of new back-ends by the implementation of CV_GridBuffer and CV_GridIndex. The class CV_BufferedGridValuesMatrix is a composition of these two classes, and should be isolated from the behavior of the back end. Further, the conceptual task of the CV_GridIndex is to divide the axes of a regular multidimensional grid up into two mutually exclusive sets. One set contains a domain axes and the other contains range axes.

TODO: Return to this after fleshing out the IIO/Multiarray2 designs in order to determine whether implementations of CV_GridIndex could benefit from some common code in an abstract class.


Buffer and Index Encapsulation Proposal
<< enumeration >>
CV_NumericType
+Number:int
+Integer:int
+Real:int
<< type >>
CV_GridDomainIndexIterator
+next():CV_GridDomainIndex
+hasNext():Boolean
+reset():void
<< type >>
CV_GridRangeIndexIterator
+next():CV_GridRangeIndex
<< type >>
CV_BufferedGridGeometry
+rank:Integer
+shape:Sequence<Integer>
+/size:Integer
<< type >>
CV_GridBuffer
+geometry:CV_BufferedGridGeometry
+type:CV_NumericType
+getNumber(idx:CV_GridIndex):Number
+getInteger(idx:CV_GridIndex):Integer
+getReal(idx:CV_GridIndex):Real
+setNumber(idx:CV_GridIndex):void
+setInteger(idx:CV_GridIndex):void
+setReal(idx:CV_GridIndex):void
+ buffer
1
1
<< type >>
CV_GridValuesMatrix
+values:Sequence<Record>
+sequencingRule:CV_SequenceRule
+startSequence:CV_GridCoordinates
<< type >>
CV_BufferedGridValuesMatrix
+getRecord(coords:CV_GridCoordinates):Record
+getRecordAttributeNumber(coords:CV_GridCoordinates,attr:CharacterString):Number
+getRecordAttributeInteger(coords:CV_GridCoordinates,attr:CharacterString):Integer
+getRecordAttributeReal(coords:CV_GridCoordinates,attr:CharacterString):Real
+setRecord(coords:CV_GridCoordinates,val:Record):void
+setRecordAttributeNumber(coords:CV_GridCoordinates,attr:CharacterString,val:Number):void
+setRecordAttributeInteger(coords:CV_GridCoordinates,attr:CharacterString,val:Integer):void
+setRecordAttributeReal(coords:CV_GridCoordinates,attr:CharacterString,val:Real):void
1
1
+ index
<< type >>
CV_GridDomainIndex
+bufferGeometry:CV_BufferedGridGeometry
+indexGeometry:CV_BufferedGridGeometry
+axes:Sequence<Integer>
+index:Sequence<Integer>
+iterator():CV_GridDomainIndexIterator
0..1
+ domainIdx
1
<< type >>
CV_GridIndex
+geometry:CV_BufferedGridGeometry
+/element:int
+setIndex(index:Sequence<Integer>):CV_GridIndex
+setIndex(domain:CV_GridDomainIndex,range:CV_GridRangeIndex):CV_GridIndex
+setDomainIndex(domain:CV_GridDomainIndex,clearRange:Boolean=True):CV_GridIndex
+setRangeIndex(range:CV_GridRangeIndex):CV_GridIndex
+setRangeIndex(attribute:CharacterString):CV_GridIndex
+clearRange():void
+isExhaustive():Boolean
0..1
1
+ rangeIdx
<< type >>
CV_GridRangeIndex
+attributeList:Dictionary<CharacterString,Sequence<Integer>>
+undefined:Set<Sequence<Integer>>
+/exhaustive:Boolean
+/attributeOrder:List<CharacterString>
+iterator():CV_GridRangeIndexIterator

Note that the implementation of GridBuffer does not have to be a memory buffer. It just requires a definition of rank and shape.

The getXXX and setXXX types may all be changed to represent the range of available numeric data types on the target system.

Describes the geometry of the n-Dimensional matrix and calculates the total size in terms of elements.

Encapsulates the coordination of a Grid Index and a Grid Buffer. Allows clients read/write access to individual record elements or entire records. Understands how to compose a record from the individual record elements.

Encapsulates the axes used for the domain. Provides an iterator over just these axes.

Encapsulates the combination of domain and range indicies as an index into the grid. Coordinates the construction of domain and range indicies such that they are mutually exclusive.

Encapsulates knowledge of which axis (or axes) belong to the range. Knows how to map attributes to specific indices on the range axes. Informs user in what order the attributes are returned, if they only index by domain. Allows users to iterate this index over the range attributes.

Created with Poseidon for UML Community Edition. Not for Commercial Use.

Figure 2: ISO 19123 extension to handle a multidimensional grid broken into domain and range axes subsets.

5.1 Common Gridded Coverage Code

5.2 J2SE Image IO Back End Implementation

The Image IO framework is depicted in Figure 3. This model must be adapted to the coverage extension shown in Figure 2. Both figures, however, have a similar structure which promises to make the implementation as straightforward as possible. The DataBuffer class is similar to the CV_GridBuffer, and SampleModel is similar to CV_GridIndex. The major differences are on the CV_GridIndex/SampleModel side, as the extension to ISO19123 assumes that each “band” of a “pixel” occupies a grid cell unto itself. The ImageIO framework allows for this possibility, and also allows for the case where bands are “packed” into one or more grid cells.

The easy part of this implementation will be to adapt “DataBuffer” and “ComponentSampleModel”. The challenging part will be to adapt the SinglePixelPackedSampleModel and MultiPixelPackedSampleModel.

This code is collected into the package: org.geotools.coverage.qgrid.iio.

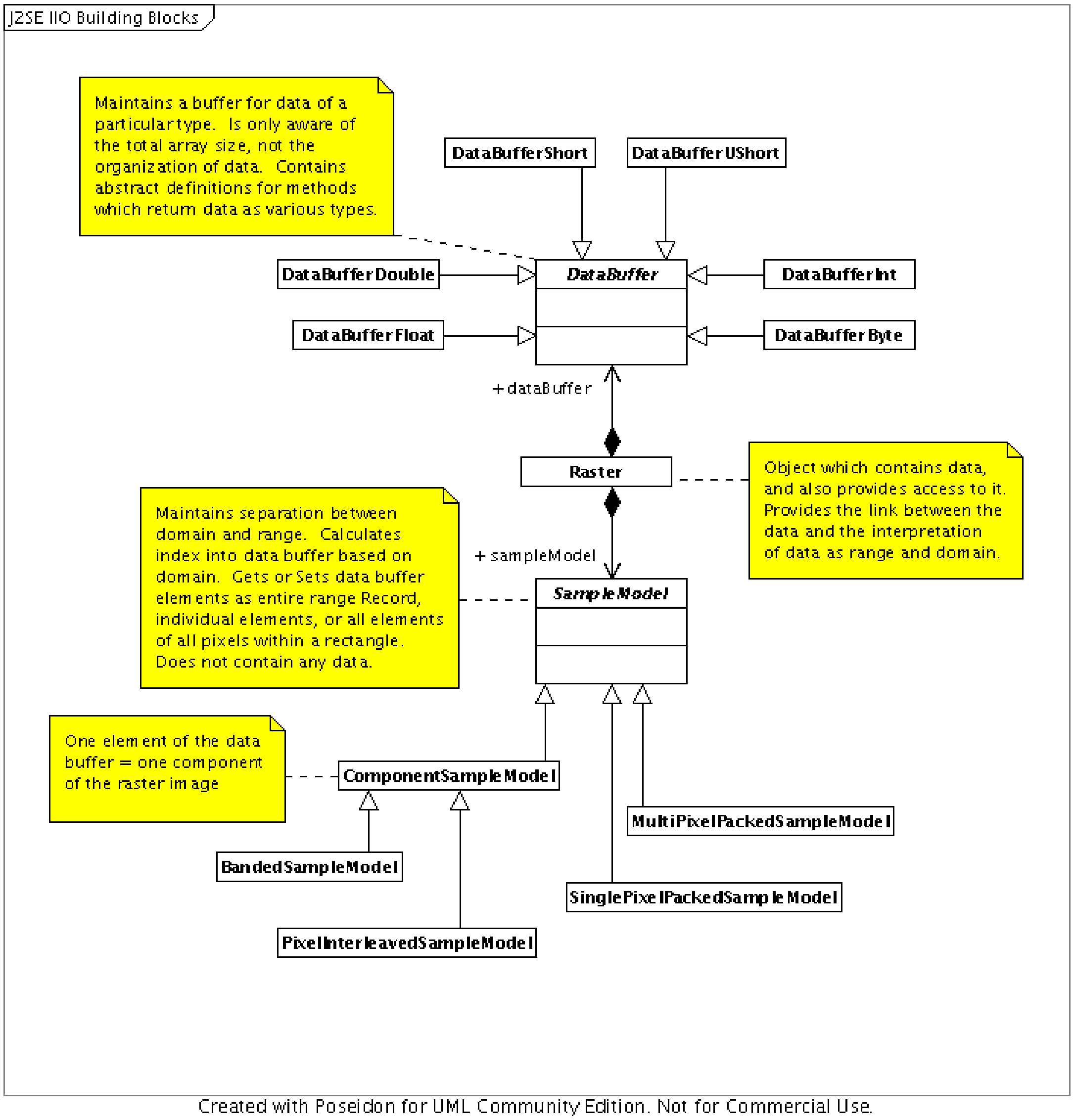
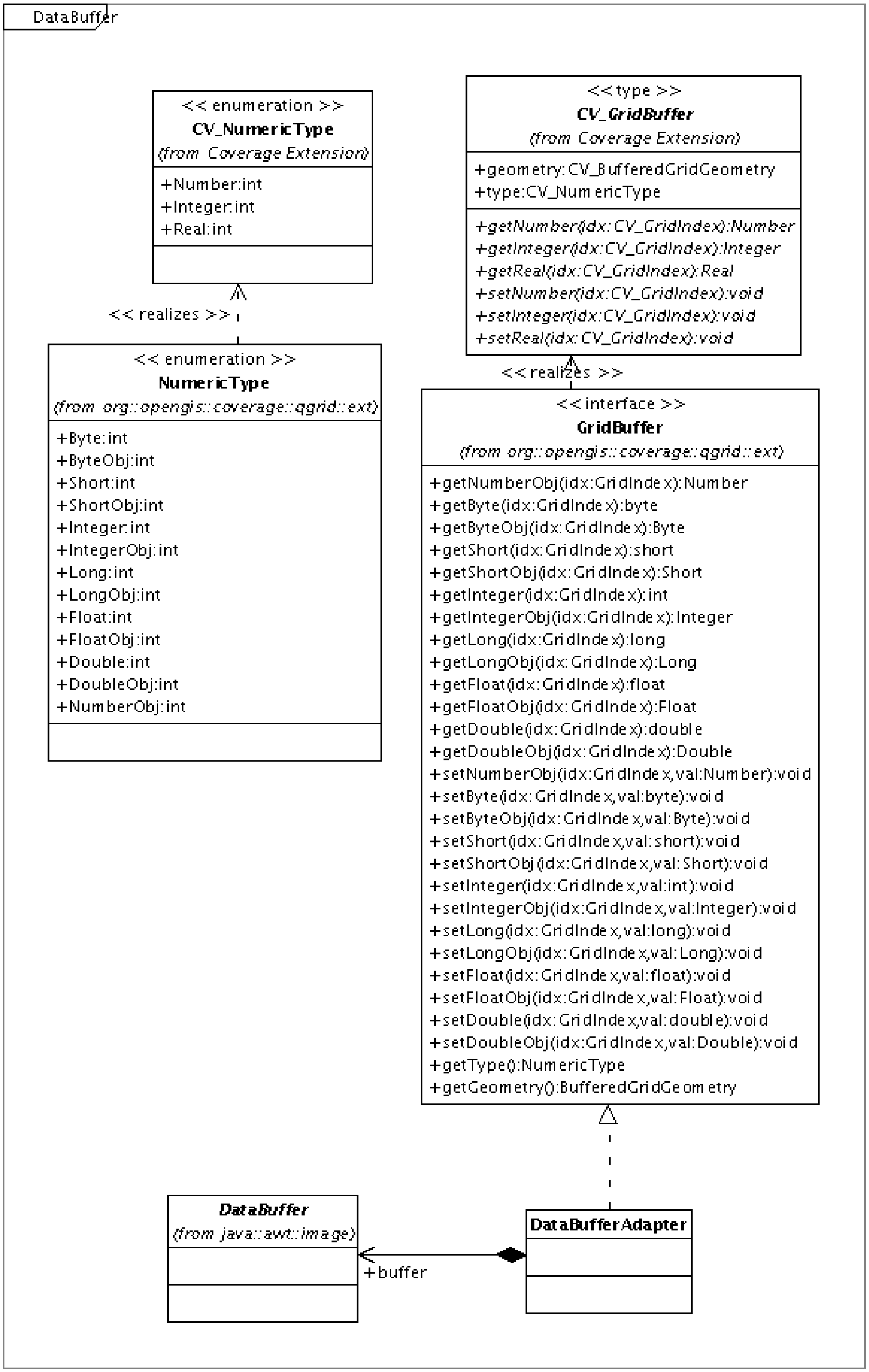

J2SE IIO Building Blocks
Maintains a buffer for data of a particular type. Is only aware of the total array size, not the organization of data. Contains abstract definitions for methods which return data as various types.
DataBufferShort
DataBufferUShort
DataBufferDouble
DataBuffer
DataBufferInt
DataBufferFloat
DataBufferByte
+dataBuffer
Raster
Object which contains data, and also provides access to it. Provides the link between the data and the interpretation of data as range and domain.
Maintains separation between domain and range. Calculates index into data buffer based on domain. Gets or Sets data buffer elements as entire range Record, individual elements, or all elements of all pixels within a rectangle. Does not contain any data.
+sampleModel
SampleModel
One element of the data buffer = one component of the raster image
ComponentSampleModel
MultiPixelPackedSampleModel
BandedSampleModel
SinglePixelPackedSampleModel
PixelInterleavedSampleModel
Created with Poseidon for UML Community Edition. Not for Commercial Use.

Figure 3: Illustration of the J2SE ImageIO framework.

5.2.1 DataBuffer

Implementation of the data buffer is accomplished by using the adapter pattern. The DataBuffer class is composed with an adapter class which implements the GridBuffer interface, as shown in Figure 4. This figure also shows the supported data types on the GeoAPI/GeoTools platform. Notice that numeric data are available as either primitive data types or numeric objects.


DataBuffer
<< enumeration >>
CV_NumericType
(from Coverage Extension)
+Number:int
+Integer:int
+Real:int
<< realizes >>
<< enumeration >>
NumericType
(from org::opengis::coverage::qgrid::ext)
+Byte:int
+ByteObj:int
+Short:int
+ShortObj:int
+Integer:int
+IntegerObj:int
+Long:int
+LongObj:int
+Float:int
+FloatObj:int
+Double:int
+DoubleObj:int
+NumberObj:int
<< type >>
CV_GridBuffer
(from Coverage Extension)
+geometry:CV_BufferedGridGeometry
+type:CV_NumericType
+getNumber(idx:CV_GridIndex):Number
+getInteger(idx:CV_GridIndex):Integer
+getReal(idx:CV_GridIndex):Real
+setNumber(idx:CV_GridIndex):void
+setInteger(idx:CV_GridIndex):void
+setReal(idx:CV_GridIndex):void
<< realizes >>
<< interface >>
GridBuffer
(from org::opengis::coverage::qgrid::ext)
+getNumberObj(idx:GridIndex):Number
+getByte(idx:GridIndex):byte
+getByteObj(idx:GridIndex):Byte
+getShort(idx:GridIndex):short
+getShortObj(idx:GridIndex):Short
+getInteger(idx:GridIndex):int
+getIntegerObj(idx:GridIndex):Integer
+getLong(idx:GridIndex):long
+getLongObj(idx:GridIndex):Long
+getFloat(idx:GridIndex):float
+getFloatObj(idx:GridIndex):Float
+getDouble(idx:GridIndex):double
+getDoubleObj(idx:GridIndex):Double
+setNumberObj(idx:GridIndex,val:Number):void
+setByte(idx:GridIndex,val:byte):void
+setByteObj(idx:GridIndex,val:Byte):void
+setShort(idx:GridIndex,val:short):void
+setShortObj(idx:GridIndex,val:Short):void
+setInteger(idx:GridIndex,val:int):void
+setIntegerObj(idx:GridIndex,val:Integer):void
+setLong(idx:GridIndex,val:long):void
+setLongObj(idx:GridIndex,val:Long):void
+setFloat(idx:GridIndex,val:float):void
+setFloatObj(idx:GridIndex,val:Float):void
+setDouble(idx:GridIndex,val:double):void
+setDoubleObj(idx:GridIndex,val:Double):void
+getType():NumericType
+getGeometry():BufferedGridGeometry
DataBuffer
(from java::awt::image)
+buffer
DataBufferAdapter

Created with Poseidon for UML Community Edition. Not for Commercial Use.

Figure 4: Adaptor pattern used when wrapping java.awt.image.DataBuffer.

TODO: Finish DataBuffer example.

TODO: Consider putting a GridBuffer abstract class which interacts with numeric objects in the “common” area.

5.3 Multiarray2 Back End Implementation

5.4 Two Dimensional Slicing with ImageIO Back End

5.4.1 Horizontal Slice

5.4.2 Vertical Slice

5.4.3 Unaligned Slice

6 Continuous Coverages

6.1 Thiessen Polygon Coverage

POSTPONE: Implementation of the Thiessen Polygon coverage is deferred to a future date.

6.2 Continuous Quadrilateral Grid Coverage

6.3 Hexagonal Grid Coverage

POSTPONE: Implementation of the Hexagonal Grid Coverage is deferred to a future date.

6.4 Triangulated Irregular Network Coverage

POSTPONE: Implementation of a TIN Coverage is deferred to a future date.

6.5 Segmented Curve Coverage

POSTPONE: Implementation of a segmented curve coverage is deferred to a future date.

6.6 Analytical Coverage

TODO: A basic structure to facilitate the creation of analytical coverages shall be identified and implemented.