Chaco Data Model

Introduction

The simplest plotting system does not have a "data model" beyond a simple list of arrays to be plotted along with some range coordinates. This is actually sufficient for a large number of plotting applications, but Chaco requires a much richer data model because it allows the construction of plotting applications with complex data interactions. Chaco's data model adds a few fundamental facets to simple static data:

1. **view**: The common notions of "range" are really narrow instances of a much more general concept, namely, a bounded view into data space. Once the notion of a "data view" is created, a lot of other concepts fall into place. Interactive plots commonly share a view or aspects of a view, and interactions with plots commonly involve manipulating the view.
2. **dimensionality**: Attaching dimensionality information to individual sequences of data establishes a common type framework for actors in the data model to exchange information in a dynamic way, without a priori knowledge of each others' capabilities.
3. **visualization pipeline**: The data model is concerned with the manipulation of data so that it can be appropriately displayed by a visual element. Most data manipulation is some sort of filtering, and the elements of the data model fit into a pipeline that sends data forwards from data space into screen space and services requests to map screen space back into data space.

Class Overview

The core classes of the data model fall roughly into roles that are defined directly by the aspect of interactive plotting described above. This is not a complete class hierarchy but it outlines the major classes.

	Base classes	Concrete classes
View-related	DataView	
DataSource	DataSource	
	AbstractDataSeries	ScalarData, PointData, ImageData
	AbstractFilter	ViewFilter, MaskFilter, CompoundFilter
	AbstractDataMap	AffineMap, PolarMap
PlotData	AbstractPlotData	XYData, MultiXYData
		ImageData, ImageData2D
		PointData, PointData2D

Class Descriptions

DataSource

DataSource is an abstract interface that must be supported by all classes in the plotting pipeline which act as providers of data. The primary subclasses of DataSource are DataSeries and Filters. For the most part, a DataSource looks like an array of values with an optional mask and metadata. (The mask is a binary array of equal length to the data array, and the metadata is a dictionary keyed on string values. Some string values are reserved and have standardized value types.)

In addition to producing an indexed array of values, a DataSource must also be able to reverse-map a value point to an index. It can do this at one of three levels of granularity:

1. return a reference to the object containing the value point
2. return a tuple of the lower and upper indices that contain the value point
3. return a single index representing the position of the value point in its array

A DataSource can be used directly as input to PlotData objects, which are then rendered by PlotRenderers. In general, however, there should be a ViewFilter placed between the DataSource and the PlotData to allow for adjustment to the range of displayed values.

Domain classes which need to be adapted to provide data to Chaco should implement DataSource. In some cases, it might be easier to subclasses one of the concrete DataSeries (ScalarData, PointData, etc.) and override the necessary functions. These classes also serve as examples of how to properly implement the DataSource interface.

AbstractFilter

Filters are subclasses of DataSource that require an input in order to generate their output data.

DataView and ViewFilter

A DataView represents a "window" into data space. It is represented by a range and has metadata just like a DataSource. DataViews do not interact directly with DataSources; rather, a DataView can produce a ViewFilter, which can then be inserted into a pipeline. Any changes to the ViewFilter are proxied back up to its parent DataView.

The simplest plotting pipelines will have one DataView and one ViewFilter per DataSource. However, by sharing DataViews, more sophisticated pipelines can easily present a coherent multi-plot display of related data.

AbstractDataSeries (subclass of DataSource)

DataSeries implement the DataSource interface and are the primary representation of numerical data in Chaco. The dimensionalities of DataSeries are:

1. **ScalarDataSeries**: An array of scalars.
   Example: `[-0.5, -0.3, 1.1, 2.3, 2.7]`
2. **PointDataSeries**: An array of 2D points. Useful for representing 2D data where there is no regular sampling or it is inconvenient to separate the components into two 1D arrays.
   Example: `[(0.1, -3.2), (0.6, -1.0), (0.87, 0.3), (1.2, 0.8)]`

3. **ImageDataSeries** (1D values): A 2D matrix of scalars. This is used as an optimized representation for bitmapped, grayscale image data.
   Example: `[ [0.8, 1.3, 10.9, 12.7],`
   `[2.2, 2.7, 6.5, 7.4],`
   `[1.7, 4.4, 3.6, 3.8] ]`
4. **ImageDataSeries** (2D values): A 2D matrix of vectors (vector map)
   Example: `[ [(1.2, 0.3), (-2.2, 2.3), (-0.4, -0.6)],`
   `[(3.0, 2.5), (1.5, 1.8), (-0.6, 1.7)] ]`
5. **ImageDataSeries** (3D/4D values): A color RGB or RGBA image.
6. **TextArray**: A linear array of text strings.
   Example: `[ "Trial 5", "Trial 6", "Trial 8" ]`

Note (In the example above, the value at [0][0] is 0.8, which cannot be used as an (x,y) coordinate.) Thus, a 1D ImageDataSeries must be indexed using two ScalarDataSeries.

AbstractPlotData

PlotData compose multiple DataSeries into a single space that can then be rendered by a PlotRenderer. By designating a certain DataSeries as an "index" and other DataSeries as "values", the PlotData defines the relationship between those spaces. Fundamentally, a PlotData maps an index into the index DataSeries to a corresponding index in the value DataSeries and passes this tuple (index_value, value_value) to the Renderer.

The table below shows the different kinds of plots defined by pairing index and value DataSeries of different cardinalities. Although "2x Scalar" is not a DataSeries, it serves as a compact representation of a uniform Image2D where the value at each (i,j) cell is just the X,Y coordinates into the corresponding two scalar DataSeries.

Index type	Value type	Plot type
Scalar	Scalar	XY, line, polar, bar/column
Point	Scalar	3D curve, contour line
	Point	vector lines
2x Scalar	Image 1D	grayscale image
	Image 2D	uniform vector map
	Image 3D	RGB picture
	Image 4D	RGBA picture
Image 2D	Image 1D	Non-uniform grayscale image
	Image 2D	Non-uniform vector map
	Image 3D	Warped RGB picture
	Image 4D	Warped RGBA picture

Note that it is not meaningful to plot a value type against an index type of fewer dimensions; in general there are no useful maps from index_n to value_ij. Furthermore, ImageDataSeries with 1D values are only used as values and not as indices because the dimensionality of the value at each cell is less than the dimensionality of the DataSeries. (Although there may be some use for an Image1D index, it is rare.)