Event Display

Tools for a JAVA based single event display

David Heddle, CNU & JLAB, 2/25/09

Purpose of an event display

1. **Diagnostics, Diagnostics, Diagnostics**
2. Realistic Visualizations

The first is significantly more important than the second. The leverage is in making an event display a better diagnostic tool, not in making it more realistic.[†]

[†] The fatal attraction, however, is always to make it more realistic.

User Requirements

- Speed
  - A new event should display quickly ~few hundred ms[†]
- Ease of Use
  - options shouldn't be buried
- Easy to build/install
  - No dependencies, no third party libraries
- Stand alone mode
- Event stream hookup
- Drag 'n Drop
- Zooming/panning etc.

- Useful views (displays)
  - Info more important than realism. 3D only when necessary.
- Accumulated views
  - Display aggregation of events
- Mouse over
  - Convenient/useful point-and-read
- Plug-in for new views
- No frozen GUI. Ever.
- Configurable

[†] You can take a bit longer if you use double buffering, which will give the illusion of speed.

Software Requirements

- OOP
  - It's the 21st century
- Platform agnostic
  - No more ignoring 90% of the world's computers
- Extensible
  - Add views without recompiling (plug-ins)
- Standards
  - "Standard is better than better" (e.g., XML)
- Double buffering

- Maintainable
  - No macros, native calls, generated code, embedding, etc
- SOA aware
  - (consumer & producer)
- SOA fault tolerant
  - Work when services are unreachable
- Multithreaded
  - Non-GUI thread notifies GUI for repaint

Technology Selections

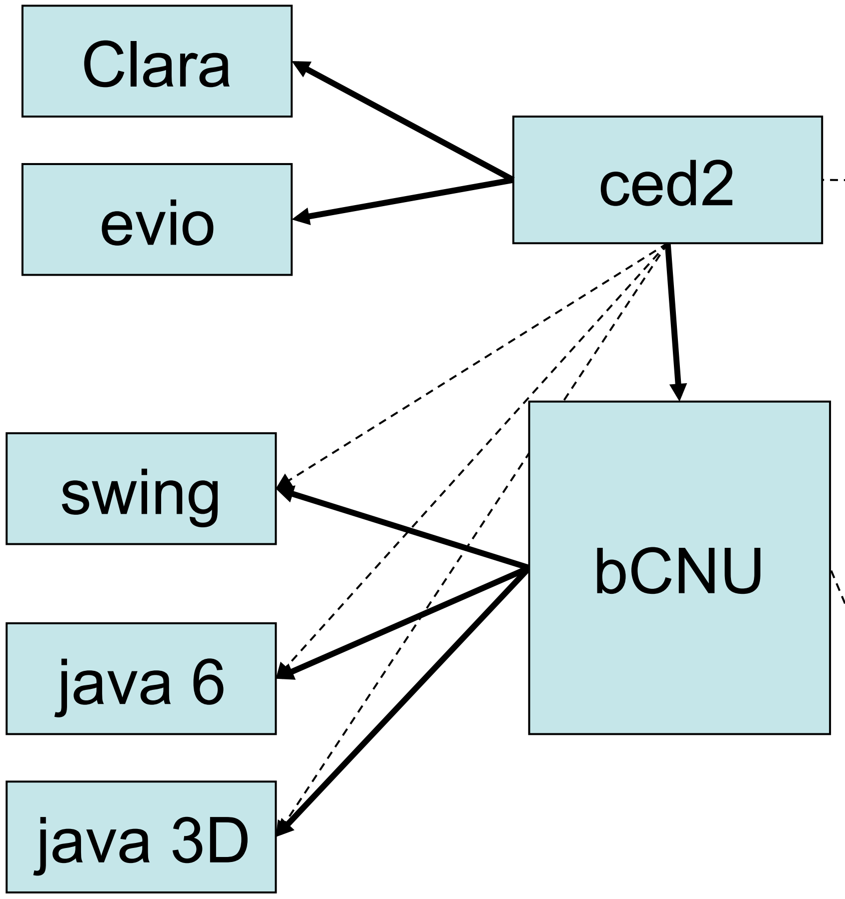
- JAVA 6
- Swing, Java 3D (calling Open GL)
- Eclipse for IDE
- Ant for build (rarely needed)
- Jars for distribution
- XML for (most) data exchange
- Layer based drawing

- Subversion for revision control
- Multiple Document Interface (MDI) (desktop & internal frames)
- Interface-rich code
- Generic attribute editing (minimize dialogs)
- Heads-up Display (preserve real-estate)

Some proprietary dependencies

- Clara API for web services
- evio & CODA common event format

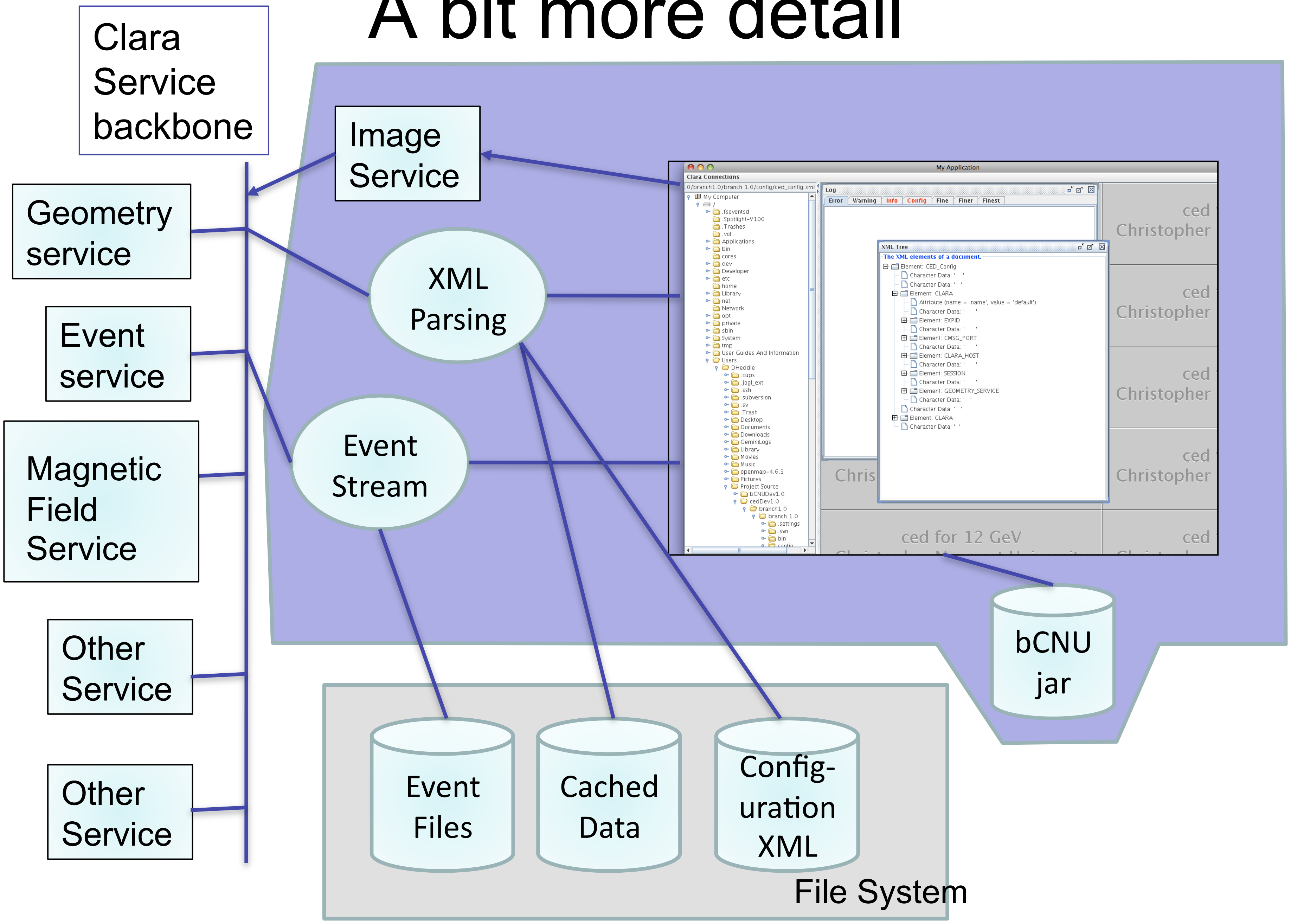
Highest level architecture



ced2 knows CLAS geometry and events. It knows nothing about screen coordinates. This piece could be replaced by other event displays (e.g, “ded”, or other applications.

bCNU handles all the app infrastructure, world ↔ local transformations, polygon rendering, rubber-banding, DnD, etc. *It knows nothing about any physics detector*. *Or Clara. Or evio. Or CODA.*

A bit more detail



.0/branch1.0/branch 1.0/config/ced_config.xml

```
My Computer
  /
    .fseventsd
    .Spotlight-V100
    .Trashes
    .vol
    Applications
    bin
    cores
    dev
    Developer
    etc
    home
    Library
    net
    Network
    opt
    private
    sbin
    System
    tmp
    User Guides And Information
    Users
      DHeddle
        .cups
        .jogl_ext
        .ssh
        .subversion
        .sv
        .Trash
        Desktop
        Documents
        Downloads
        GeminiLogs
        Library
        Movies
        Music
        openmap-4.6.3
        Pictures
        Project Source
          bCNUDev1.0
          cedDev1.0
            branch1.0
              branch 1.0
                .settings
                .svn
                bin
                config
```

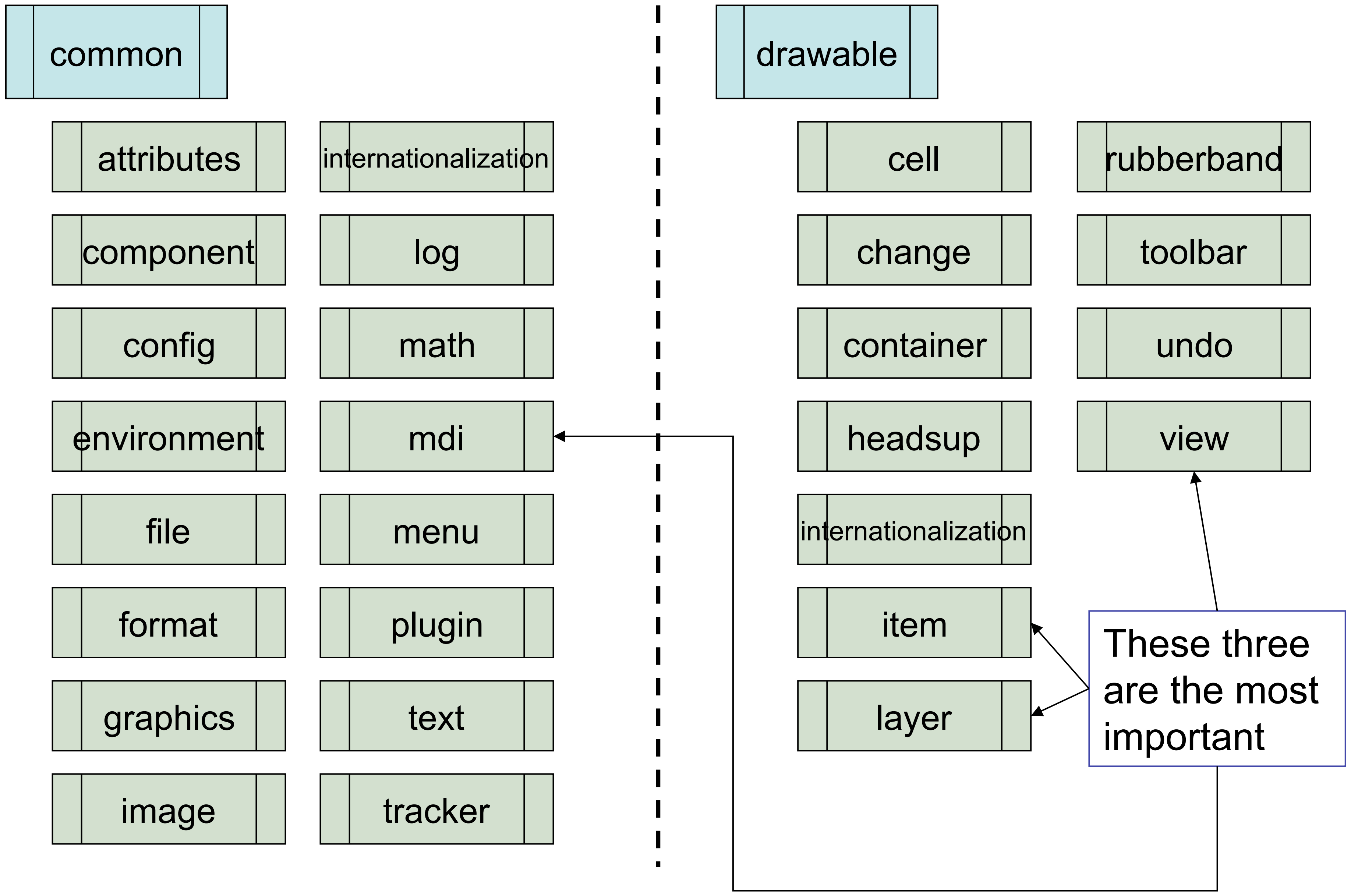
The XML elements of a document.

```
Element: CED_Config
  Character Data: '  '
  Character Data: '  '
  Element: CLARA
    Attribute (name = 'name', value = 'default')
    Character Data: '    '
    Element: EXPID
    Character Data: '    '
    Element: CMSG_PORT
    Character Data: '    '
    Element: CLARA_HOST
    Character Data: '    '
    Element: SESSION
    Character Data: '    '
    Element: GEOMETRY_SERVICE
    Character Data: '  '
  Character Data: '  '
  Element: CLARA
  Character Data: ' '
```

In a nutshell

- ced2 reads the geometry and creates world-based Items (bCNU objects)
- The Items get placed on z-ordered Layers
- The Layers get drawn from front to back
- The set of Layers may be a simple as:
  - Detector
  - Magnetic Field
  - Event
  - Annotation

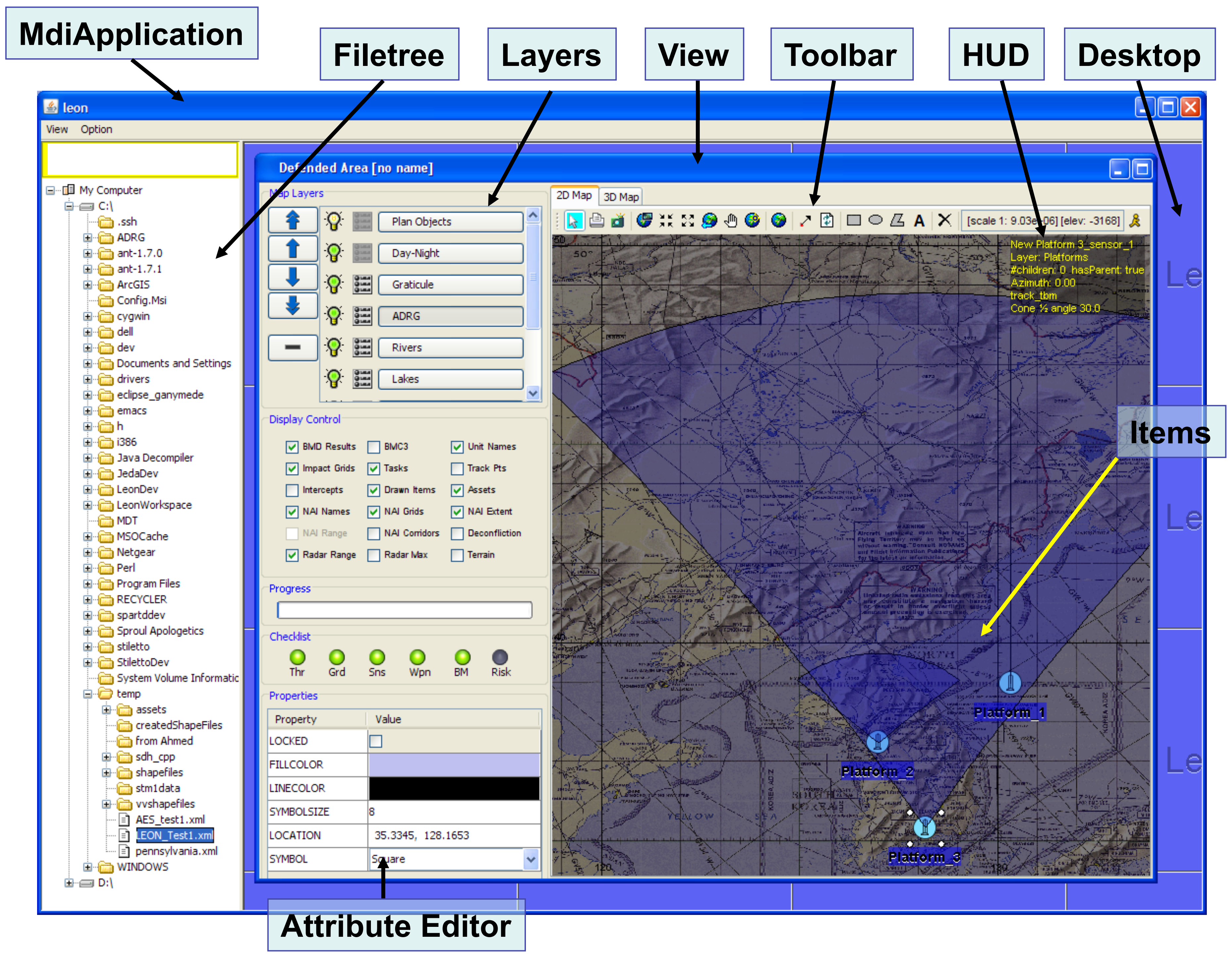
bCNU package Structure



To use this package

1. You Instantiate a *MdiApplication*, which creates a desktop.
2. You design *views* (what do I want to display?) Each view is an internal frame.
3. You design *layers* (you can put everything on one layer—or put everything on its own layer—but optimal is ~handful of layers.)
4. You implement *items* for the objects to be rendered, add them to a layer, fill them with data (model.)†

† bCNU "more or less" adheres to the Model-View-Controller (MVC) paradigm. As with all paradigms, practical considerations sometimes make it impossible to live up to the ideal.



[scale 1: 9.03e-06] [elev: -3168]

New Platform 3_sensor_1
Layer: Platforms
#children: 0 hasParent: true
Azimuth: 0.00
track_tbm
Cone ½ angle 30.0

✓ BMD Results	☐ BMC3	✓ Unit Names
✓ Impact Grids	✓ Tasks	☐ Track Pts
☐ Intercepts	✓ Drawn Items	✓ Assets
✓ NAI Names	✓ NAI Grids	✓ NAI Extent
☐ NAI Range	☐ NAI Corridors	☐ Deconfliction
✓ Radar Range	☐ Radar Max	☐ Terrain

Property	Value
LOCKED	☐
FILLCOLOR	
LINECOLOR	■
SYMBOLSIZE	8
LOCATION	35.3345, 128.1653
SYMBOL	Square

1) Creating a MdiApplication

```
/**
 * Create a MdiApplication Multiple Document Interface).
 *
 * @param keyVals an optional variable length list of attributes in
 *            type-value pairs. For example, AttributeType.TITLE, "my
 *            application", AttributeType.CENTER, true, etc.
 */
public MdiApplication(Object... keyVals) {
```

The construct **Object... keyVals** represents a variable length argument list. This is common throughout bCNU. The **keyVals** are (*name, value*) pairs. The name is a String, and the value is any Java object. These are collectively known as *attributes*, although *properties* would have been a better name. This mechanism has two huge advantages:

1) Arbitrary user-data can be supplied using the same signature.

2) The attribute editing mechanism can be used to reduce the number of dialogs required.

Example (class Ced2 extends MdiApplication)

```java
/**
 * Main program used for testing only.
 *
 * @param args the command line arguments.
 */
public static void main(String[] args) {
       Ced2 frame = new Ced2(AttributeType.NAME, "ced2",
       AttributeType.CENTER, true,
       AttributeType.CONFIGFILE, "ced_config.xml",
       AttributeType.FRACTION, 0.95,
       AttributeType.FILETREE, true,
       AttributeType.TILE, true,
       AttributeType.LOGVIEW, true,
       AttributeType.XMLTREEVIEW, true,
       AttributeType.TILESTRING, "ced for 12 GeV\nCNU");

       frame.setVisible(true);
}
```

2) Creating Views

```java
/**
 * Create a base 2D view,
 * @param desktop the desktop that will hold this view.
 * @param keyVals the variable length list of attributes.
 */
public BaseView2D(JDesktopPane desktop, Object... keyVals) {
```

The desktop is accessible from the MdiApplication object
Below we create a StView, which extends a BaseView2D.

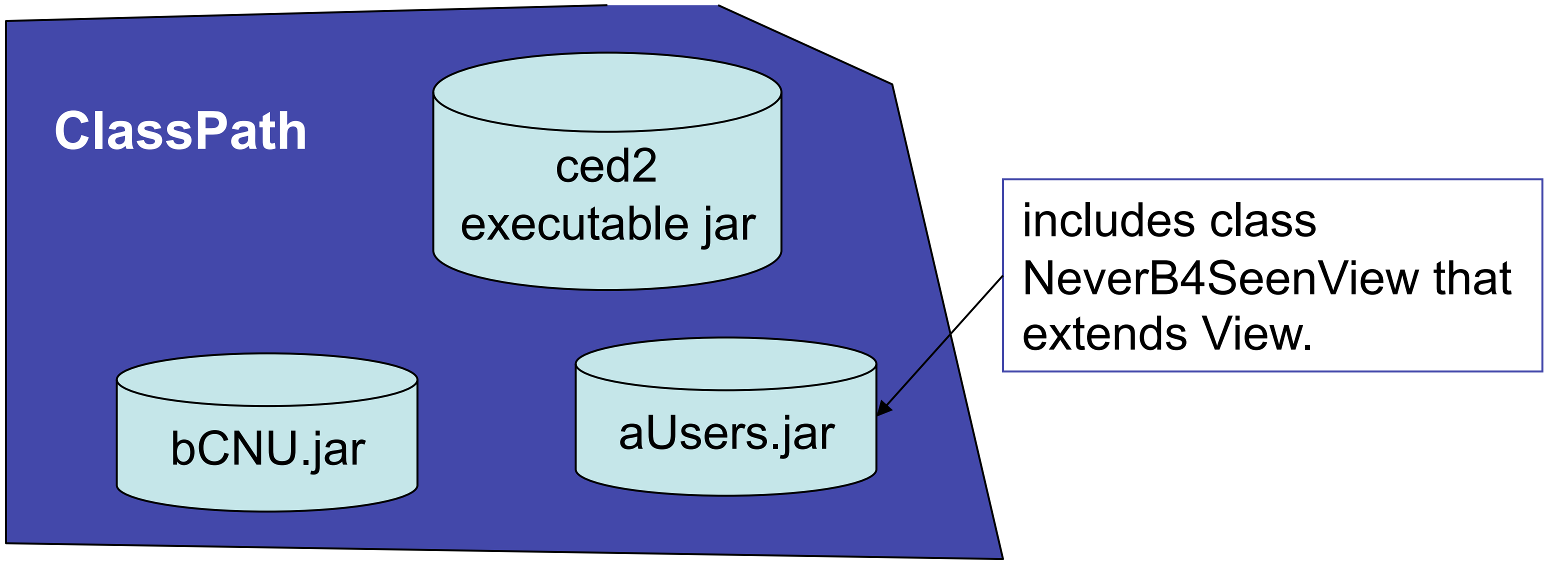
```java
StView stview = new StView(ced2.getDesktop(),
      AttributeType.NAME, "Start Counter View",
      AttributeType.VISIBLE, true,
      AttributeType.BACKGROUND, new Color(64, 64, 64, 128),
      "My Attribute", myDataObject,
```

Every view will contain a layerDatabase that is a collection of its layers. There is one default layer: *AnnotationLayer*.

Plugins: another way to create views (using reflection)

In a nutshell, at startup[†]:

1. Scan all classes in ClassPath (a regexp filter is optional)
2. For any class that extends View, invoke the static method getInstance(). This creates one instance.



[†]In principle it could be done in a timer while running—so a new view could be dropped in the ClassPath and *viola*! it pops up.

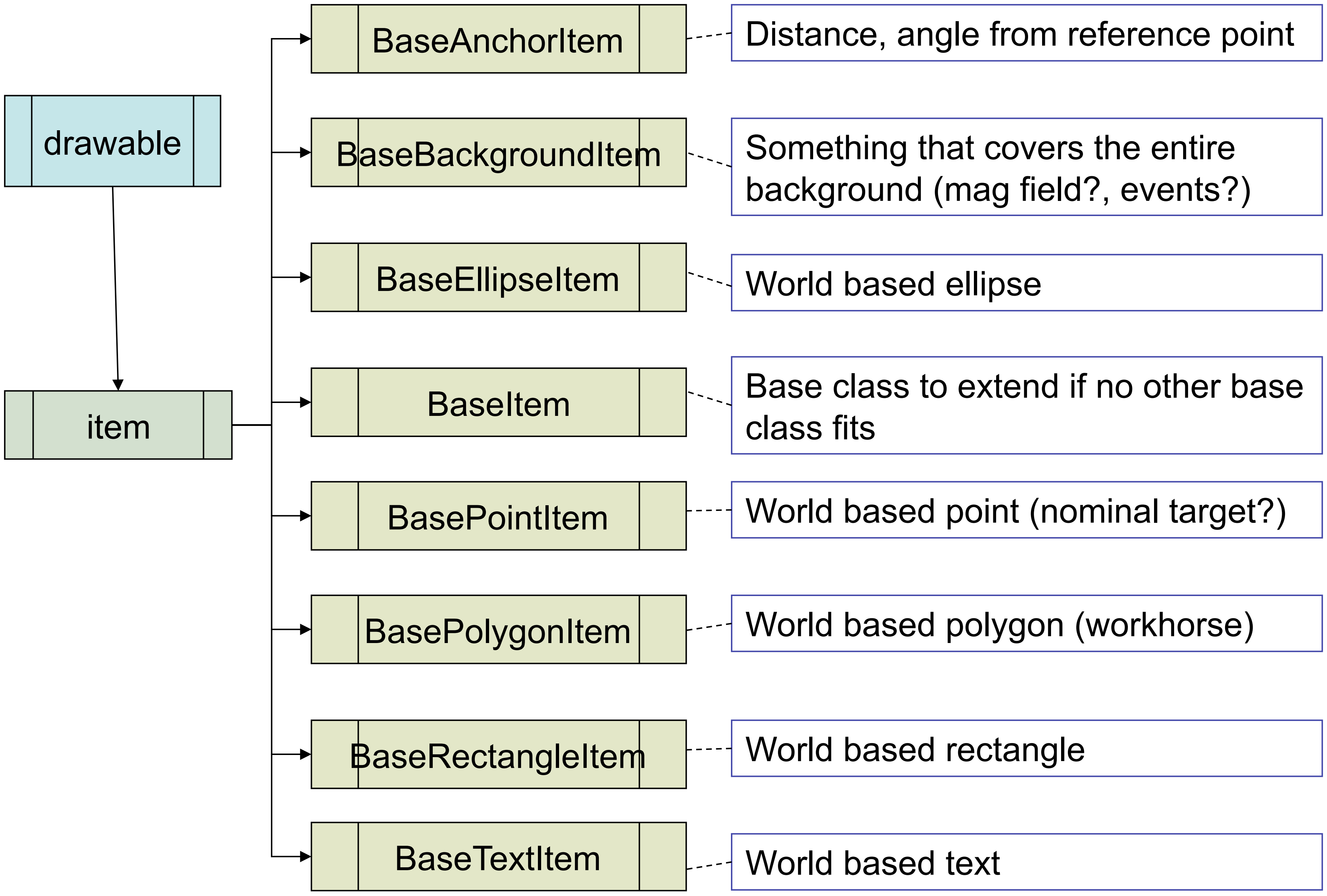
Creating Layers

```
/**
 * Creates a BaseLayer with a given name.
 * @param layerDatabase the collection of layers for a view.
 * @param name the name of the layer.
 */

 public BaseLayer(LayerDatabase layerDatabase, String name) {
```

example

```
BaseLayer detectorLayer = new BaseLayer(
       stView.getLayerDatabase(), "Detector Layer");

BaseLayer eventLayer = new BaseLayer(
       stView.getLayerDatabase(), "Event Layer");
```

Creating Items—the meat & potatoes


drawable
item
BaseAnchorItem
BaseBackgroundItem
BaseEllipseItem
BaseItem
BasePointItem
BasePolygonItem
BaseRectangleItem
BaseTextItem
Distance, angle from reference point
Something that covers the entire background (mag field?, events?)
World based ellipse
Base class to extend if no other base class fits
World based point (nominal target?)
World based polygon (workhorse)
World based rectangle
World based text

Example

The polygon annotation tool rubberbands a screen polygon. Those points are converted to world coordinates. The collection of world coordinates is used to create a BasePolygonItem.

"this" must implement IConverter, which means it can convert world ↔ local

```
/**
 * Create a polygon item, probably from a rubberband.
 * Use all default attributes.
 * @param pp the screen coordinates of the vertices.
 * @return the new polygon item.
 */
public BaseItem createPolygonItem(Point pp[]) {
            WorldPolygon wpoly = new WorldPolygon(this, pp);
            BaseLayer layer =
      layerDatabase.getOrCreateLayer(AnnotationLayerName);

            return new BasePolygonItem(layer,
                  ItemAttributeType.WORLDPOLYGON, wpoly,
                  ItemAttributeType.ROTATABLE, true,
                  ItemAttributeType.TYPE, "Annotation",
                  ItemAttributeType.RESIZABLE, true);
}
```

For a detector frame, something like

```
public BaseItem createDetectorFrame(String detectorName) {

//ask GeomtryService for the frame vertices
           WorldPolygon wpoly =
GeometryService.getFrameVertices(detectorName);

//get (or create, if necessary) the detector layer
           BaseLayer layer =
      layerDatabase.getOrCreateLayer("DetectorLayer");

//create the item
           return new BasePolygonItem(layer,
                 ItemAttributeType.WORLDPOLYGON, wpoly,
                 ItemAttributeType.ROTATABLE, false,
                 ItemAttributeType.FILLCOLOR,
Color.gray,
                 ItemAttributeType.LINECOLOR, Color.red,
                 ItemAttributeType.RESIZABLE, false);
}
```

In practice, the BasePolygonItem is extended

```
public class DriftChamber extends BasePolygonItem {

public DriftChamber (BaseLayer layer,
                          Object... keyVals
      super(layer, keyVals);
}

/**
 * The method where the custom drawing occurs. Draw only
 * the item, not its children.
 * @param g the Graphics context.
 */
@Override
protected void customDraw(Graphics g){
      //draw the cells within the frame
```

The frame vertices and wire positions were passed in the attribute list.

A primer on how the drawing is performed

A JComponent (BaseContainer) is placed in the “business” part of each view. Its paintComponent method is a loop over all layers, and each layer’s draw method is a loop over all its items (which then loop over their children.)

```
@Override
public void paintComponent(Graphics g) {
        for (BaseLayer layer : layerDatabase) {
                if (layer.isVisible() {
                        layer.draw(g);
                }
        }
}
```

Layer class’s draw method

```
public void draw(Graphics g) {
        for (Item item : items) {
                if (item.isVisible() {
                        item.draw(g);
                }
        }
}
```

More complicated under the hood

- An “offscreen” Item that is drawn will take as much CPU as if it were visible. Also, its “pick” check can be expensive. Thus care is taken to identify out-of-play Items[†].
  - Items are asked for their outlines, which are checked for intersection with the clip region (the region being repainted.)
  - An invisible grid is imposed on the drawing area. Each cell maintains a list of items that it intersects. When picking the cell is determined, and only items in the cell (in reverse order) are checked.

[†] In older systems (e.g., X-Windows) another check could be made: don’t draw Items that are occluded by other items. This no longer works, because of the widespread use of transparency.

If I made it this far…

- Nobody is as surprised as I am. That’s enough for one talk.