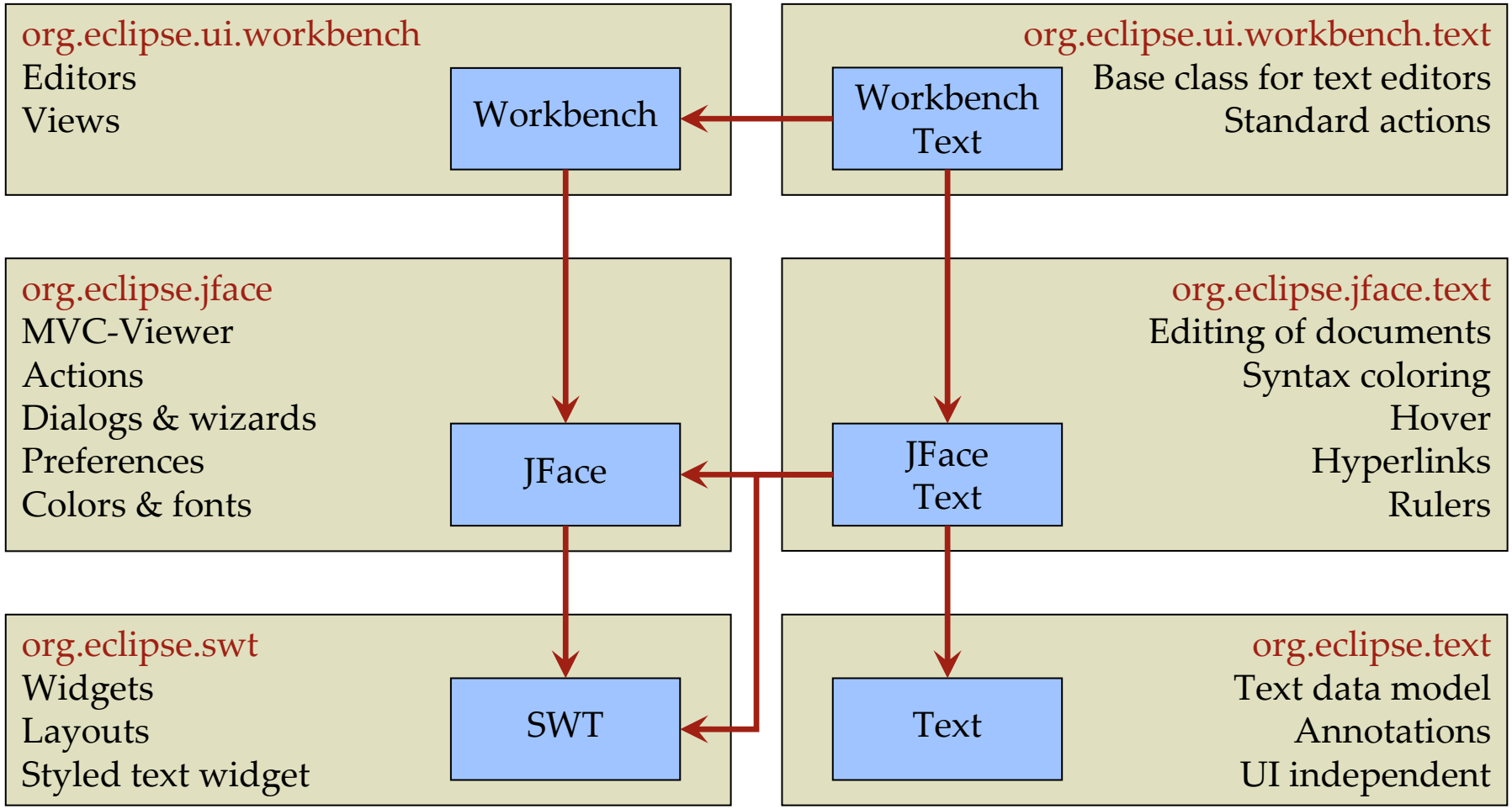
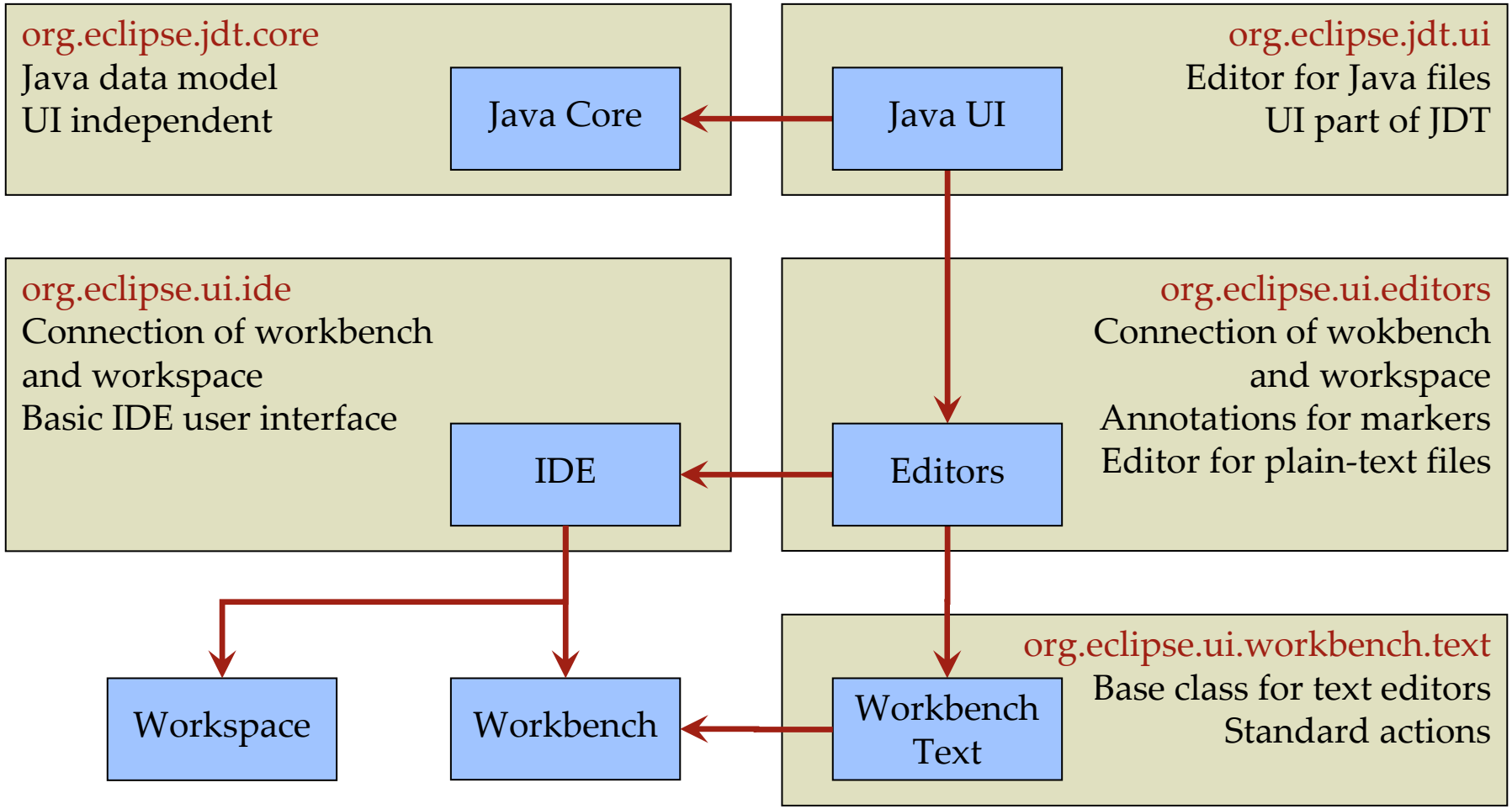
Text Editor


- Eclipse contains a flexible text editor framework
  - Used e.g. for Java editor
- Layered architecture
  - SWT component for styled text
  - UI-independent text data model
  - JFace viewer for styled text
  - Workbench integration for text editors
  - Workspace integration for editing text files
  - Language-specific editors such as Java editor
- Custom text editor can build on each layer
  - Basic decision: depend on workspace or not
  - RCP applications do not use workspace
    - Some limitations and rough edges
    - Cutting line seems arbitrary for some parts
    - Old monolithic design still visible

SWT, JFace & Workbench



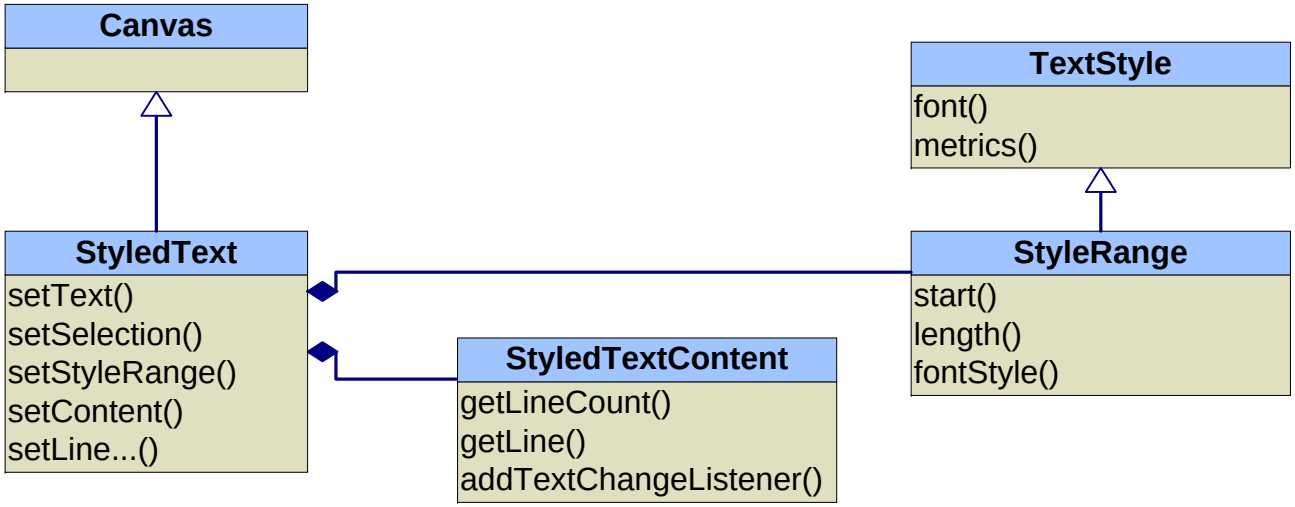
Workbench & Workspace



SWT Styled Text

- Custom SWT widget
  - Does not use a native widget
    - Especially not the windows RTF widget
  - Supports text with font styles
  - Line alignment and basic paragraph formatting
    - Line bullets
    - Tabulators
- Limitations
  - Focus lies on editing source code
    - One font for the whole text
    - No non-text parts
  - Font, images, ... supported, but complicated
  - Code snippets on SWT homepage

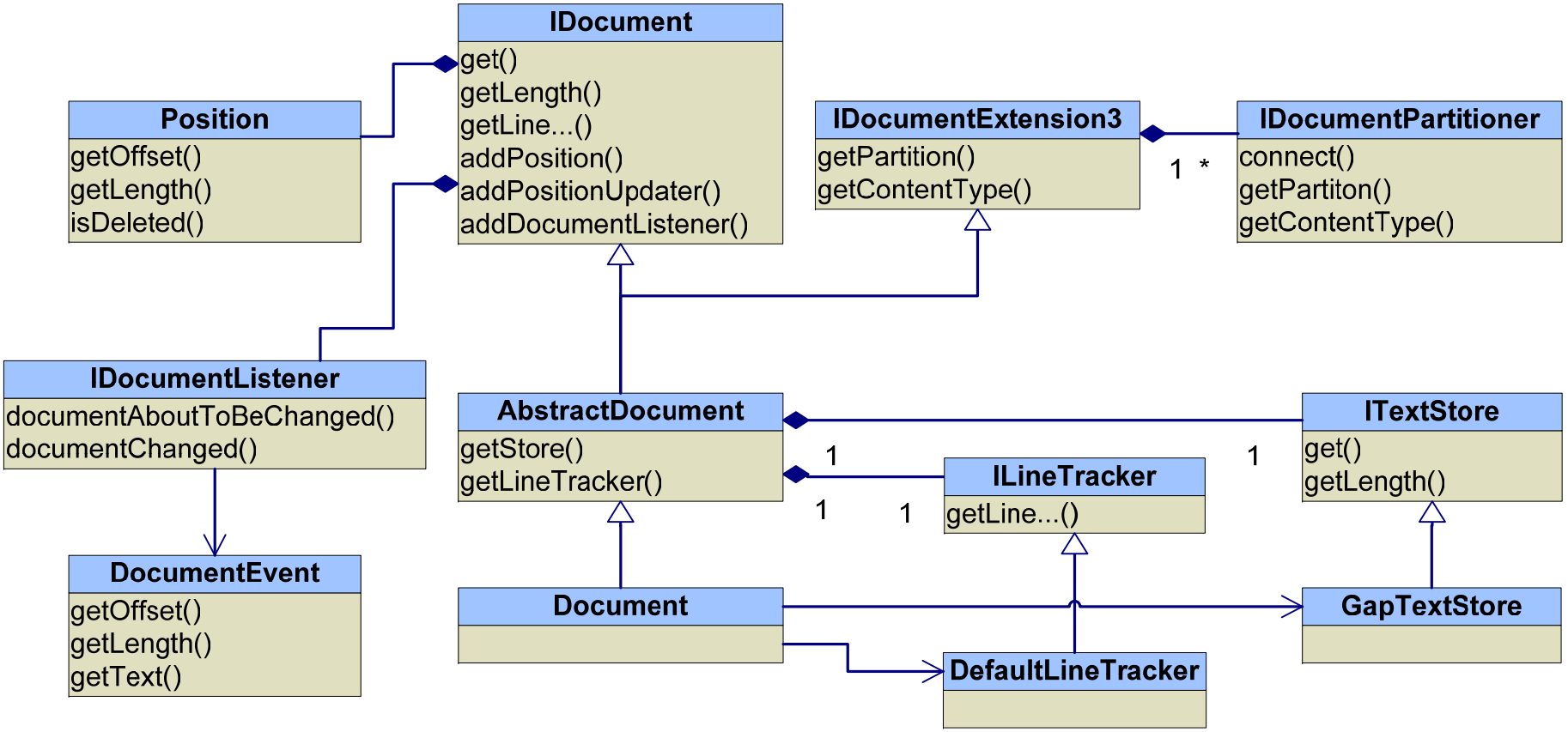
SWT Styled Text



Text Model


- UI independent
  - No dependency on SWT or JFace
- Data model for text documents
  - Text content
    - IDocument with default implementation
  - Text storage
    - Implementation of gap text storage
  - Positions
    - Updated when text is modified
  - Annotations
    - Additional non-text information bound to a position
  - Partitions
    - Separates a document in different content types
    - Example: Java source code and comments

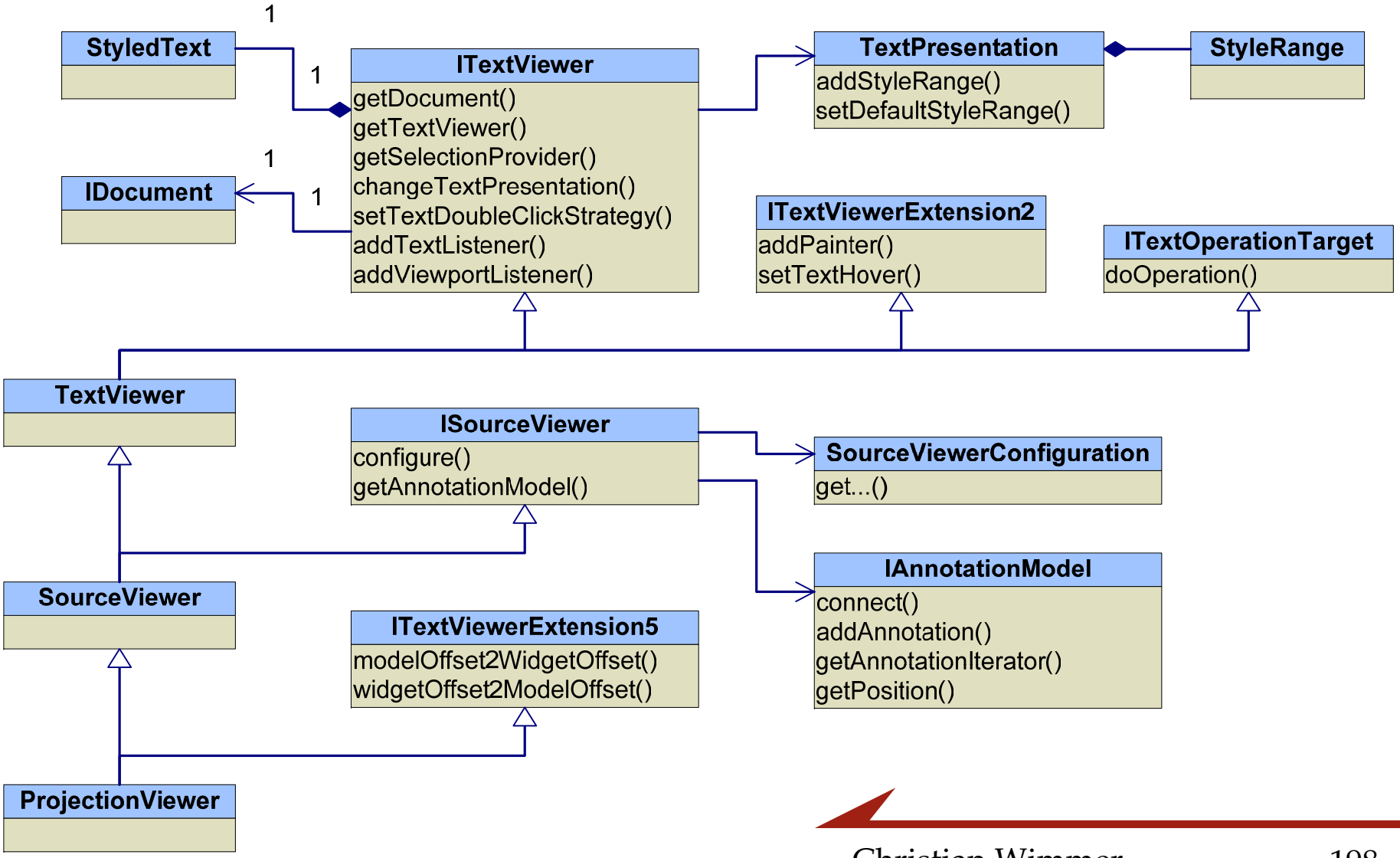
Text Model


IDocument
get()
getLength()
getLine...()
addPosition()
addPositionUpdater()
addDocumentListener()
Position
getOffset()
getLength()
isDeleted()
IDocumentExtension3
getPartition()
getContentType()
1 *
IDocumentPartitioner
connect()
getPartiton()
getContentType()
IDocumentListener
documentAboutToBeChanged()
documentChanged()
AbstractDocument
getStore()
getLineTracker()
1
1
1
ILineTracker
getLine...()
1
ITextStore
get()
getLength()
DocumentEvent
getOffset()
getLength()
getText()
Document
DefaultLineTracker
GapTextStore

Text Viewer


- Text Viewer
  - Shows an IDocument in a StyledText
- Source Viewer
  - Adds support for annotations
  - Adds source code features to text viewer
  - Configuration via separate SourceViewerConfiguration object
    - Presentation reconciler
    - Reconciler
    - Text hover
    - Hyperlink detector and presenter
    - Double click strategy
    - Content assistant
    - Content formatter
- Projection Viewer
  - Adds support for folding
  - Folding regions are added as annotations

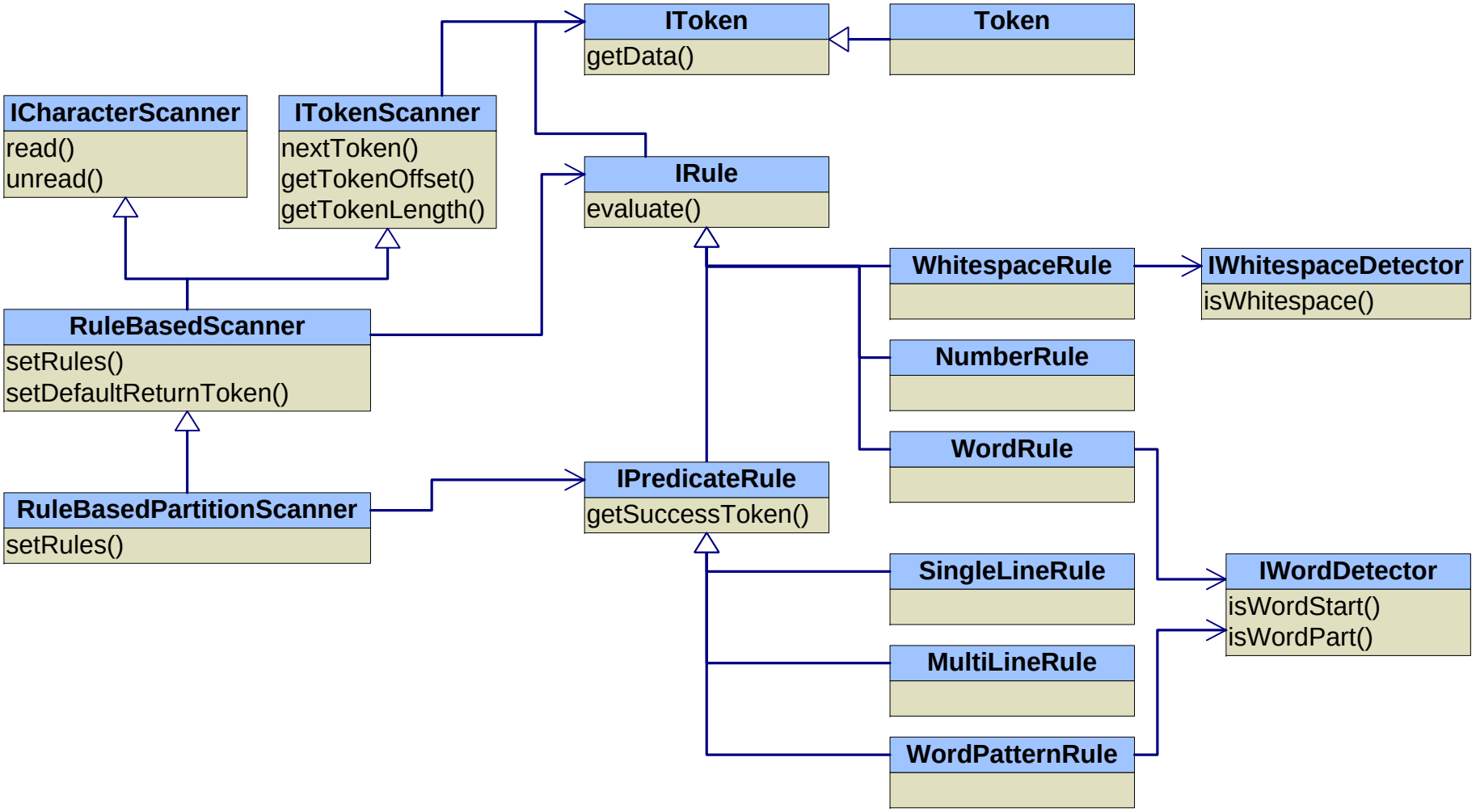
Text Viewer


StyledText
1
1
ITextViewer
getDocument()
getTextViewer()
getSelectionProvider()
changeTextPresentation()
setTextDoubleClickStrategy()
addTextListener()
addViewportListener()
TextPresentation
addStyleRange()
setDefaultStyleRange()
StyleRange
1
1
IDocument
ITextViewerExtension2
addPainter()
setTextHover()
ITextOperationTarget
doOperation()
TextViewer
ISourceViewer
configure()
getAnnotationModel()
SourceViewerConfiguration
get...()
IAnnotationModel
connect()
addAnnotation()
getAnnotationIterator()
getPosition()
SourceViewer
ITextViewerExtension5
modelOffset2WidgetOffset()
widgetOffset2ModelOffset()
ProjectionViewer

Scanner & Rules

- Scanner
  - Split a text into tokens
  - Token can carry arbitrary data
    - String that identifies a content type
    - “TextAttribute” for syntax highlighting
  - Default implementation: rule based scanner
    - Sufficient for many scanner tasks
    - Scanner has a list of rules
    - The first matching rule wins and returns its token
- Rules
  - Useful implementations available
    - WordRule: Return tokens for different keywords
  - Predicate rules
    - Return one token if the rule matches
    - SingleLineRule, MultiLineRule

Scanner & Rules



Scanner & Rules


```
TextAttribute nameAttr = new TextAttribute(...);
Token nameToken = new Token(tagAttr);

WordRule nameRule = new WordRule(nameDetector, nameToken);
nameRule.addWord("drawing", tagToken);
nameRule.addWord("width", attributeToken);
...

Token valueToken = new Token(valueAttr);
valueRule = new SingleLineRule("\"", "\"", valueToken);

Token commentToken = new Token(commentAttr);
commentRule = new MultiLineRule("<!--", "-->", commentToken, (char) 0, true);
```

Rule for xml names

Unknown names

tag names

attribute names

Rule for xml values

Rule for xml commments

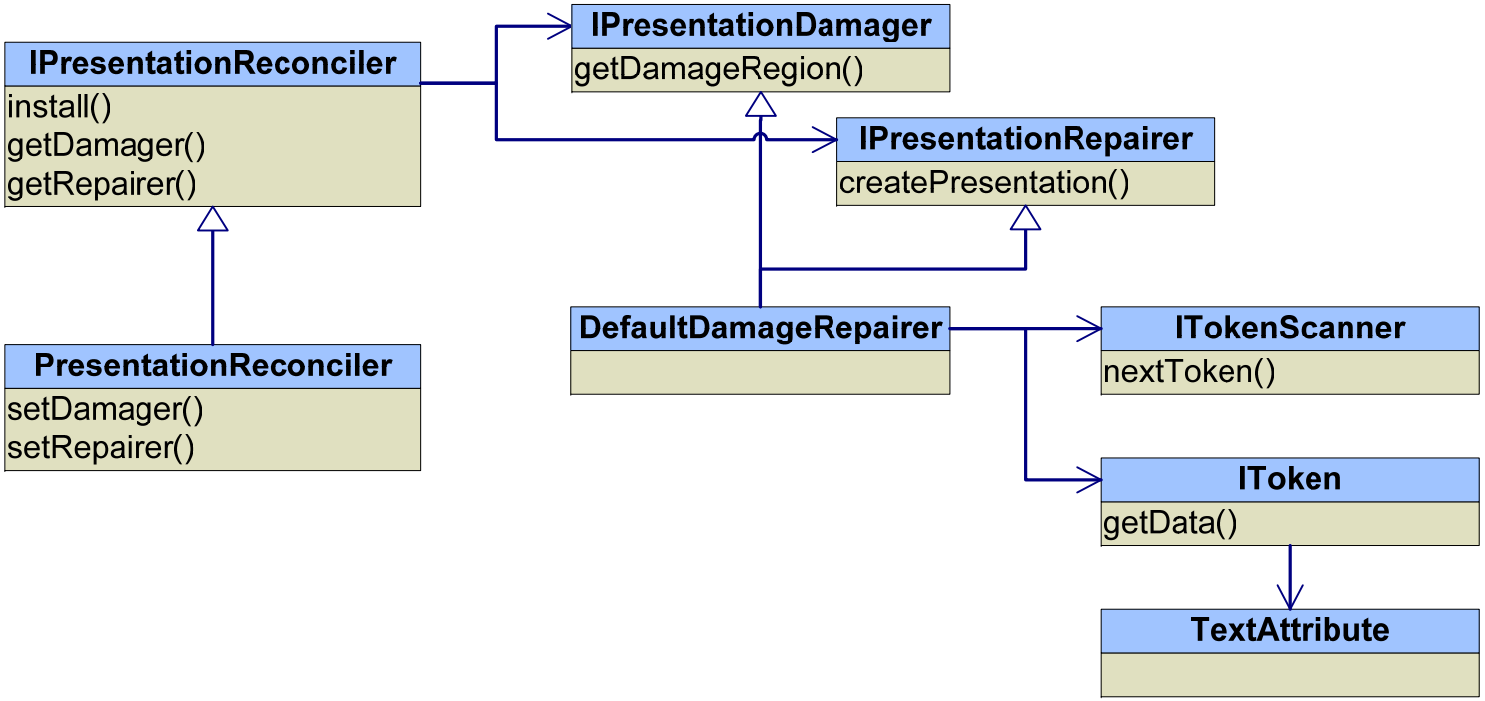
Word detector for xml names

```
private IWordDetector nameDetector = new IWordDetector() {
  public boolean isWordStart(char c) {
    return Character.isLetter(c) || c == '_' || c == ':';
  }
  public boolean isWordPart(char c) {
    return isWordStart(c) || Character.isDigit(c) || c == '.' || c == '-';
  }
};
```

Syntax Coloring


- IPresentationReconciler
  - Add-on for text viewer
  - Configured in SourceViewerConfiguration
  - Installs itself as an add-on on a text viewer
    - Reduces the complexity of the text viewer
    - General pattern for most advanced editor features
- Damage-repair strategy
  - Syntax coloring must be updated when text changes
  - IPresentationDamager determines range
  - IPresentationRepairer computes new text attributes
  - Separate objects for each content type
- Default implementation with rule based scanner
  - Only rules must be specified
  - Tokens carry text attribute in data property

Syntax Coloring



Syntax Coloring

Configuration of a PresentationReconciler

```
setDocumentPartitioning(DrawingPartitions.PARTITIONING);


RuleBasedScanner contentScanner = new RuleBasedScanner();
contentScanner.setRules(new IRule[] { nameRule, valueRule });

DefaultDamagerRepairer contentDR = new DefaultDamagerRepairer(contentScanner);
setDamager(contentDR, IDocument.DEFAULT_CONTENT_TYPE);
setRepairer(contentDR, IDocument.DEFAULT_CONTENT_TYPE);

RuleBasedScanner commentScanner = new RuleBasedScanner();
commentScanner.setDefaultReturnToken(commentToken);

DefaultDamagerRepairer commentDR = new DefaultDamagerRepairer(commentScanner);
setDamager(commentDR, DrawingPartitions.COMMENT);
setRepairer(commentDR, DrawingPartitions.COMMENT);
```

Specify partitioning

Define scanner with rules

Use scanner for DR

Install DR for a content type

Comments have only one text attribute

```
public class DrawingSourceViewerConfiguration ... {
  public IPresentationReconciler getPresentationReconciler(ISourceViewer sourceViewer) {
    return new DrawingPresentationReconciler(...);
  }
}
```

Subclass used here to encapsulate code

Synchronize Text with Data Model


- User expects immediate feedback during typing
  - Text structure in outline view
  - Graphical preview
  - Detection of syntax errors
- Text must be analyzed during typing
  - Must not delay typing
    - Only when no change for some time
    - Background thread that does not block UI
  - Must not crash on syntax errors or incomplete text
- Strategies
  - Completely re-build model after each change
    - Simple to implement
  - Analyze changes and modify model
    - Scales for large models

Synchronize Text with Data Model


- IReconciler
  - IReconcilingStrategy for each content type
    - With extension interface
  - Utility class MonoReconciler
    - One strategy for the whole document
- Example
  - Manage a Drawing object for outline and thumbnail view
  - Manage position of <figure>-tags for folding and navigation
- Where to store the model objects
  - Text viewer is available (nearly) everywhere
  - Use your own subclass
    - Cast to subclass when need to access model objects

Synchronize Text with Data Model


```
public class DrawingReconcilingStrategy implements
                IReconcilingStrategy, IReconcilingStrategyExtension {
  public void initialReconcile() {
    buildFigureTags();
    buildFolding();

    XMLBinding.loadDrawing(viewer.getDrawing(),
        new StringReader(document.get()));
  }

  public void reconcile(IRegion partition) {
    initialReconcile();
  }
}
```

Detect figure-tags in the document

Update the folding structure of the text viewer

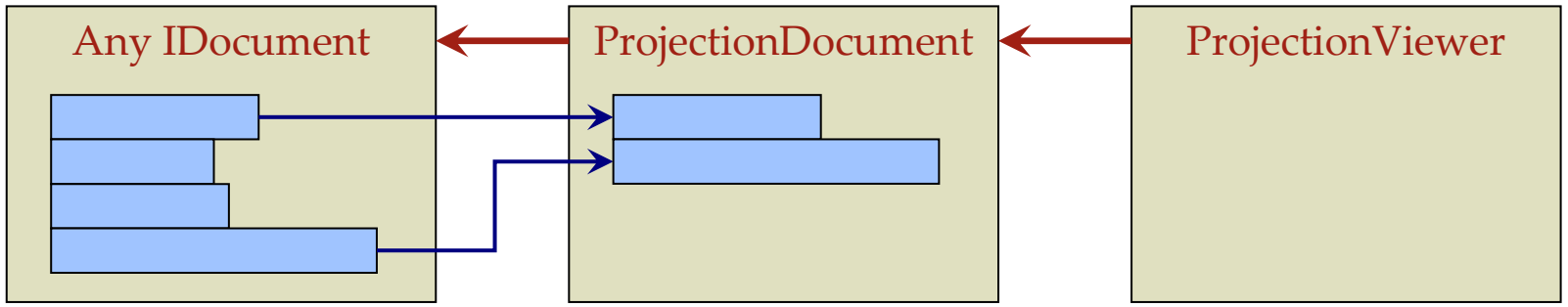
Update the model used for the outline and thumbnail

No difference between first and subsequent reconciles

```
public class DrawingSourceViewerConfiguration ... {
  public IReconciler getReconciler(ISourceViewer sourceViewer) {
    IReconcilingStrategy strategy = new DrawingReconcilingStrategy(...);
    return new MonoReconciler(strategy, false);
  }
}
```

Folding

- Internal: additional layer between IDocument and text viewer



- Usage: Folding regions are specified as Annotations
  - Annotation class “ProjectionAnnotation”
  - Position of annotation = region that can be folded
  - Per default, the first line is shown in folded state
    - Can be changed
  - Special text viewer subclass “ProjectionViewer”
  - “ProjectionSupport” coordinates the components

Folding


```
public class DrawingTextEditor extends TextEditor {
  protected ISourceViewer createSourceViewer(Composite parent,
      IVerticalRuler ruler, int styles) {
    return new ProjectionViewer(parent, ruler, getOverviewRuler(),
        isOverviewRulerVisible(), styles);
  }

  public void createPartControl(Composite parent) {
    super.createPartControl(parent);

    ProjectionViewer viewer = (ProjectionViewer) getSourceViewer();
    ProjectionSupport projectionSupport = new ProjectionSupport(viewer,
        getAnnotationAccess(), getSharedColors());
    projectionSupport.install();

    viewer.doOperation(ProjectionViewer.TOGGLE);
  }
}
```

Text viewer subclass that supports folding

Create and install folding support

Turn folding mode on

Folding


```
public class DrawingReconcilingStrategy ... {
  private void updateFolding(Position[] positions) {

    Map<ProjectionAnnotation, Position> newAnnotations =
        new HashMap<ProjectionAnnotation, Position>();


    for (int i = 0; i < positions.length; i++) {
      ProjectionAnnotation annotation = new ProjectionAnnotation();
      newAnnotations.put(annotation, positions[i]);
    }

    viewer.getProjectionAnnotationModel().modifyAnnotations(
            oldAnnotations, newAnnotations, null);
  }
}
```

The folding regions

Positions are not stored in annotations, but in map

Add annotation with its position

Update the folding annotations of the viewer

- Simplified and incomplete fragment
  - Get oldAnnotations from viewer
  - Set collapsed state of new annotation based on matching old annotation
  - Ensure that positions are on line boundaries

Other Features


- Double click strategy
  - What to select when user performs double click
  - Usual behavior: Select an entire word
    - Word boundaries depend on language
  - Example
    - Select the whole figure tag when user double-clicks on end tag.
  - Implementation
    - Implement interface ITextDoubleClickStrategy
    - Override method getDoubleClickStrategy() in viewer configuration
- Hyperlink detector
  - What to do when user activates hyperlink
    - Usually click with pressed Ctrl-key
    - Select some text in some editor
  - Example
    - Jump to start of figure tag when clicking on end tag
    - Target is always in same editor as source

Hyperlink Detector


```
public class DrawingHyperlinkDetector implements IHyperlinkDetector {
  public IHyperlink[] detectHyperlinks(ITextViewer textViewer,
      IRegion region, boolean canShowMultipleHyperlinks) {

    DrawingSourceViewer viewer = (DrawingSourceViewer) textViewer;
    int pos = region.getOffset();

    for (Position[] tag : viewer.getFigureTags()) {
      if (tag[1].includes(pos)) {
        return new IHyperlink[] { new TagHyperlink(viewer, tag) };
      }
    }
    return null;
  }
}
```

Access to data model

Click position of user

IHyperlink implementation

```
public class DrawingSourceViewerConfiguration ... {
  public IHyperlinkDetector[] getHyperlinkDetectors(ISourceViewer sourceViewer) {
    return new IHyperlinkDetector[] { new DrawingHyperlinkDetector() };
  }
}
```

Content Assist


- IContentAssistProcessor
  - Completion proposals
    - Reasonable text fragments at a cursor position
  - Context Information
    - Tooltips shown for a cursor position
  - Activation with activation character
    - Example: “<“ for list of allowed tag names
  - Activation with Ctrl-Space
    - Requires a “ContentAssistAction” in editor
- Example
  - Completion proposals for tag- and attribute names
  - Not context sensitive
    - Always all tag- or attribute names shown

Text Editor Summary


- Eclipse text editor offers rich functionality
  - All features expected from a modern IDE
  - Focus lies on source code editors, but not limited to this
- Layered architecture
  - SWT component for styled text
    - Too low level for most usage scenarios
  - UI-independent text data model
    - Usually no need to extend it
  - JFace viewer for styled text
    - Use SourceViewer or ProjectionViewer
    - Configuration via explicit SourceViewerConfiguration
  - Workbench integration for text editors
  - Workspace integration for editing text files
    - Convenient base class for text editors: TextEditor
  - Language-specific editors such as Java editor