Coverage Analysis

File name: ECore2GMF.eol
Total number of statements: 4352
Number of executed statements: 2122
Coverage percentage: 49%

```
//import 'ECoreUtil.eol';
//import 'Formatting.eol';



-- The root EPackage
var ePackage := ECore!EPackage.all.first();

-- Create GmfTool basics
var toolRegistry := new GmfTool!ToolRegistry;
var palette := new GmfTool!Palette;
palette.title := ePackage.name + 'Palette';
toolRegistry.palette := palette;

-- Create Nodes and Links GmfTool tool groups
var nodesToolGroup := new GmfTool!ToolGroup;
nodesToolGroup.title := 'Objects';
nodesToolGroup.collapsible := true;
palette.tools.add(nodesToolGroup);

var linksToolGroup;
linksToolGroup := new GmfTool!ToolGroup;
linksToolGroup.title := 'Connections';
linksToolGroup.collapsible := true;
palette.tools.add(linksToolGroup);

-- Greate GmfGraph basics

var canvas : new GmfGraph!Canvas;
canvas.name := ePackage.name;

var figureGallery : new GmfGraph!FigureGallery;
figureGallery.name := 'Default';
canvas.figures.add(figureGallery);

-- Create GmfMap basics

var mapping : new GmfMap!Mapping;
var canvasMapping : new GmfMap!CanvasMapping;
mapping.diagram := canvasMapping;
canvasMapping.diagramCanvas := canvas;
canvasMapping.domainModel := ePackage;
canvasMapping.domainMetaElement := getDiagramClass();
canvasMapping.palette := palette;

-- Process Node EClasses

for (class in getNodes()) {

    -- Create GmfTool creation tool
    var creationTool := createCreationTool(class);
    nodesToolGroup.tools.add(creationTool);

    class.~creationTool := creationTool;

    -- Create GmfGraph figure
    var figureDescriptor := createFigureDescriptor(class.name + 'Figure');
    class.~figureDescriptor := figureDescriptor;

    -- Create GmfGraph figure
    var figure := class.createFigure();
    figureDescriptor.actualFigure := figure;
    figure.name := figureDescriptor.name;

    -- Create GmfGraph label
    var label := class.createLabel();
    label.name := class.getLabelName() + 'Figure';

    -- If the label belongs to a compartment owning node
    -- limit its vertical size
    if (not class.getCompartmentReferences().isEmpty()) {
        var maxDimension := new GmfGraph!Dimension;
        maxDimension.dx := 10000;
        maxDimension.dy := 50;
```

```
        label.maximumSize := maxDimension;
    }

    -- Create GmfGraph node
    var node := new GmfGraph!Node;
    node.name := class.name;
    node.figure := figureDescriptor;
    canvas.nodes.add(node);

    -- store for use in the compartments
    class.~diagramNode := node;

    if (class.getNodeSize().isDefined()) {
        var defaultSizeFacet := new GmfGraph!DefaultSizeFacet;
        node.facets.add(defaultSizeFacet);
        defaultSizeFacet.defaultSize := class.getNodeSize();
    }

    -- Create GmfMap node mapping
    var nodeMapping : new GmfMap!NodeMapping;
    nodeMapping.diagramNode := node;
    nodeMapping.domainMetaElement := class;
    nodeMapping.tool := creationTool;
    class.~nodeMapping := nodeMapping;

    -- Create GmfGraph diagram label
    var labelPlacement := class.getLabelPlacement();
if (labelPlacement <> 'none') {
        var diagramLabel := new GmfGraph!DiagramLabel;
        diagramLabel.name := class.getLabelName();
        diagramLabel.elementIcon := class.labelHasIcon(true);
        canvas.labels.add(diagramLabel);

        if (labelPlacement = 'internal') {
            figure.children.add(label);

            -- Create GmfGraph child access
            var childAccess : new GmfGraph!ChildAccess;
            childAccess.figure := label;
            childAccess.accessor := 'getFigure' + class.getLabelName() + 'Figure';
            figureDescriptor.accessors.add(childAccess);

            diagramLabel.accessor := childAccess;
            diagramLabel.figure := figureDescriptor;
        } else {
            -- Create GmfGraph label figure
            var labelFigureDescriptor := createFigureDescriptor(class.name + 'LabelFigure');
            labelFigureDescriptor.actualFigure := label;
            diagramLabel.figure := labelFigureDescriptor;
        }
        class.~diagramLabel := diagramLabel;

        -- Create GmfMap feature label mapping
        var featureLabelMapping : new GmfMap!FeatureLabelMapping;
        nodeMapping.labelMappings.add(featureLabelMapping);
        featureLabelMapping.diagramLabel := diagramLabel;
        featureLabelMapping.editPattern := class.getLabelEditPattern();
        featureLabelMapping.editorPattern := class.getLabelEditPattern();
        featureLabelMapping.viewPattern := class.getLabelViewPattern();
        featureLabelMapping.features.addAll(class.getLabelAttributes());
        featureLabelMapping.readOnly := class.getLabelReadOnly();
    }

    -- Process EAttributes (gmf.labels)
    for (attribute in getLabelledAttributesFor(class)) {
        -- Create GmfGraph label
        var label := new GmfGraph!Label;
        label.name := attribute.getLabelName() + 'Figure';
        label.text := attribute.getAnnotationValue('gmf.label', 'label.text');

        -- Create GmfGraph diagram label
        var diagramLabel := new GmfGraph!DiagramLabel;
        diagramLabel.name := attribute.getLabelName();
        diagramLabel.elementIcon := false;
        canvas.labels.add(diagramLabel);

        figure.children.add(label);

        -- Create GmfGraph child access
        var childAccess : new GmfGraph!ChildAccess;
        childAccess.figure := label;
        childAccess.accessor := 'getFigure' + label.name;
```

```
        figureDescriptor.accessors.add(childAccess);

        diagramLabel.accessor := childAccess;
        diagramLabel.figure := figureDescriptor;

        -- Create GmfMap feature label mapping
        var featureLabelMapping : new GmfMap!FeatureLabelMapping;
        nodeMapping.labelMappings.add(featureLabelMapping);
        featureLabelMapping.diagramLabel := diagramLabel;
        featureLabelMapping.features.add(attribute);
        featureLabelMapping.editPattern := attribute.getLabelEditPattern();
        featureLabelMapping.editorPattern := attribute.getLabelEditPattern();
        featureLabelMapping.viewPattern := attribute.getLabelViewPattern();
        featureLabelMapping.features.addAll(attribute.getLabelAttributes());
        featureLabelMapping.readOnly := attribute.getReadOnly();
    }
}

for (containment in getDiagramClass().getContainmentReferences()) {
    for (class in containment.eType.getConcreteSubtypes().select(c|c.isNode())) {

        -- Create GmfMap top node reference
        if (class.~topNodeReference.isUndefined()) {
            var topNodeReference := new GmfMap!TopNodeReference;
            mapping.nodes.add(topNodeReference);
            topNodeReference.containmentFeature := containment;
            class.~nodeMapping.~nested := true;
            topNodeReference.ownedChild := class.~nodeMapping;
            class.~topNodeReference := topNodeReference;
        }
    }
}

-- Do phantom nodes

for (phantom in getPhantomNodes()) {
    if (phantom.~topNodeReference.isUndefined()) {
        var topNodeReference := new GmfMap!TopNodeReference;
        mapping.nodes.add(topNodeReference);
        topNodeReference.ownedChild := phantom.~nodeMapping;
        phantom.~topNodeReference := topNodeReference;
    }
}

-- Do affixed references
for (class in getNodes()) {
    for (affixed in class.getAffixedReferences()) {
        for (child in getNodes().select(s|not s.abstract and (s = affixed.eType or
s.eAllSuperTypes.includes(affixed.eType)))) {
            -- Create GmfMap child reference

            var childReference := new GmfMap!ChildReference;
            class.~nodeMapping.children.add(childReference);
            childReference.containmentFeature := affixed;

            if (child.~nodeMapping.~nested.isDefined()) {
                childReference.referencedChild := child.~nodeMapping;
            }
            else {
                child.~nodeMapping.~nested := true;
                childReference.ownedChild := child.~nodeMapping;
            }

            -- Update affixed property
            child.~diagramNode.affixedParentSide := GmfGraph!Direction#NORTH;
        }
    }
}

-- Do compartment references
for (class in getNodes()) {

    for (containment in class.getCompartmentReferences()) {

        var referenceName := class.name + containment.name.firstToUpperCase();

        -- Create GmfGraph rectangle

        var figureDescriptor := class.~figureDescriptor;
        var figure := figureDescriptor.actualFigure;

        var compartmentRectangle := new GmfGraph!Rectangle;
```

```
        compartmentRectangle.outline := false;
        figure.children.add(compartmentRectangle);
        compartmentRectangle.name := referenceName + 'CompartmentFigure';

        var compartmentChildAccess := new GmfGraph!ChildAccess;
        figureDescriptor.accessors.add(compartmentChildAccess);
        compartmentChildAccess.figure := compartmentRectangle;
        compartmentChildAccess.accessor := 'get' + compartmentRectangle.name;

        -- Create GmfGraph compartment

        var compartment := new GmfGraph!Compartment;
        compartment.name := referenceName + 'Compartment';
        compartment.collapsible := containment.isCollapsible();
        compartment.figure := class.~figureDescriptor;
        canvas.compartments.add(compartment);
        compartment.accessor := compartmentChildAccess;

        -- Create GmfMap compartment mapping

        var compartmentMapping := new GmfMap!CompartmentMapping;
        compartmentMapping.compartment := compartment;
        class.~nodeMapping.compartments.add(compartmentMapping);
        class.~nodeMapping.relatedDiagrams.add(canvasMapping);

        for (child in getNodes().select(s|not s.abstract and (s = containment.eType or
s.eAllSuperTypes.includes(containment.eType)))) {

            -- Create GmfMap child reference

            var childReference := new GmfMap!ChildReference;
            class.~nodeMapping.children.add(childReference);
            childReference.compartment := compartmentMapping;
            childReference.containmentFeature := containment;

            if (child.~nodeMapping.~nested.isDefined()) {
                childReference.referencedChild := child.~nodeMapping;
            }
            else {
                child.~nodeMapping.~nested := true;
                childReference.ownedChild := child.~nodeMapping;
            }

        }
    }

}

-- Delete unused GmfMap node mappings

for (nodeMapping in GmfMap!NodeMapping.all.clone()) {
    if (nodeMapping.~nested.isUndefined()
        and (not nodeMapping.domainMetaElement.isPhantom()))
     delete nodeMapping;
}

-- Process Link EClasses

for (class in getLinks()) {
    -- Do not create links for classes that cannot be contained
    -- in the diagram or in the opposite of their sourceFeature

    if (getAllSuitableContainmentReferences(class).size() == 0) continue;

    -- Create GmfTool creation tool
    var creationTool := createCreationTool(class);
    linksToolGroup.tools.add(creationTool);

    -- Create GmfGraph figure descriptor
    var figureDescriptor := new GmfGraph!FigureDescriptor;
    figureDescriptor.name := class.name + 'Figure';
    figureGallery.descriptors.add(figureDescriptor);

    -- Create GmfGraph polyline connection
    var polylineConnection := new GmfGraph!PolylineConnection;
    figureDescriptor.actualFigure := polylineConnection;
    polylineConnection.name := figureDescriptor.name;
    polylineConnection.formatConnection(class);

    -- Create GmfGraph connection
    var connection := new GmfGraph!Connection;
    connection.name := class.name;
```

```
    connection.figure := figureDescriptor;
    canvas.connections.add(connection);

    var linkMappings : Sequence;

    for (containmentFeature in getAllSuitableContainmentReferences(class)) {
        -- Create GmfMap link mapping
        var linkMapping : new GmfMap!LinkMapping;
        mapping.links.add(linkMapping);
        linkMapping.containmentFeature := containmentFeature;
        linkMapping.diagramLink := connection;
        linkMapping.domainMetaElement := class;
        linkMapping.tool := creationTool;
        linkMapping.sourceMetaFeature := class.getLinkSourceFeature();
        linkMapping.linkMetaFeature := class.getLinkTargetFeature();
        linkMappings.add(linkMapping);

        var sourceEndConstraint := class.getSourceConstraint();
        var targetEndConstraint := class.getTargetConstraint();
        if (sourceEndConstraint.isDefined() or targetEndConstraint.isDefined()) {
            linkMapping.creationConstraints := new GmfMap!LinkConstraints;
            if (sourceEndConstraint.isDefined()) {
                linkMapping.creationConstraints.sourceEnd := new GmfMap!Constraint;
                linkMapping.creationConstraints.sourceEnd.body := sourceEndConstraint;
            }
            if (targetEndConstraint.isDefined()) {
                linkMapping.creationConstraints.targetEnd := new GmfMap!Constraint;
                linkMapping.creationConstraints.targetEnd.body := targetEndConstraint;
            }
        }
    }

    -- Create connection label
    var hasLabel := not class.getLabelAttributes().isEmpty();

    if (hasLabel) {
        var labelFigureDescriptor := createFigureDescriptor(class.getLabelName() + 'Figure');
        var label := class.createLabel();
        label.name := class.getLabelName() + 'Label';
        labelFigureDescriptor.actualFigure := label;

        var diagramLabel := new GmfGraph!DiagramLabel;
        diagramLabel.figure := labelFigureDescriptor;
        diagramLabel.name := class.getLabelName();
        diagramLabel.elementIcon := class.labelHasIcon(false);
        canvas.labels.add(diagramLabel);

        var featureLabelMapping := new GmfMap!FeatureLabelMapping;
        featureLabelMapping.diagramLabel := diagramLabel;
        featureLabelMapping.editPattern := class.getLabelEditPattern();
        featureLabelMapping.editorPattern := class.getLabelEditPattern();
        featureLabelMapping.viewPattern := class.getLabelViewPattern();
        featureLabelMapping.features.addAll(class.getLabelAttributes());
        featureLabelMapping.readOnly := class.getLabelReadOnly();
        for (linkMapping in linkMappings) {
            linkMapping.labelMappings.add(featureLabelMapping);
        }
    }

}

for (reference in getReferenceLinks()) {
    var referenceName := reference.getLongName();

    -- Create GmfTool creation tool
    var creationTool := createCreationTool(reference);
    linksToolGroup.tools.add(creationTool);

    -- Create GmfGraph figure descriptor
    var figureDescriptor := createFigureDescriptor(referenceName + 'Figure');

    -- Create GmfGraph external label figure descriptor
    var externalLabelFigureDescriptor := createFigureDescriptor(referenceName + 'ExternalLabelFigure');

    -- Greate GmfGraph external label
    var externalLabel := new GmfGraph!Label;
    externalLabel.name := referenceName + 'ExternalLabel';
    externalLabel.text := reference.getLinkLabel();
    externalLabelFigureDescriptor.actualFigure := externalLabel;

    -- Create GmfGraph external label diagram label
    var externalDiagramLabel := new GmfGraph!DiagramLabel;
```

```
    externalDiagramLabel.figure := externalLabelFigureDescriptor;
    externalDiagramLabel.name := externalLabel.name;
    externalDiagramLabel.elementIcon := false;
    canvas.labels.add(externalDiagramLabel);

    -- Create GmfGraph polyline connection
    var polylineConnection := new GmfGraph!PolylineConnection;
    figureDescriptor.actualFigure := polylineConnection;
    polylineConnection.name := figureDescriptor.name;

    -- Create GmfGraph connection
    var connection := new GmfGraph!Connection;
    connection.name := referenceName;
    connection.figure := figureDescriptor;
    canvas.connections.add(connection);

    polylineConnection.formatConnection(reference);

    -- Create GmfMap link mapping
    var linkMapping : new GmfMap!LinkMapping;
    mapping.links.add(linkMapping);
    linkMapping.diagramLink := connection;
    linkMapping.linkMetaFeature := reference;
    linkMapping.tool := creationTool;

    -- Create GmfMap external label mapping
    var externalLabelMapping := new GmfMap!DesignLabelMapping;
    externalLabelMapping.readOnly := true;
    externalLabelMapping.diagramLabel := externalDiagramLabel;
    linkMapping.labelMappings.add(externalLabelMapping);
}

-- Order things in the GmfMap model

mapping.nodes := mapping.nodes.asSequence().sortBy(n | -n.ownedChild.domainMetaElement.eAllSuperTypes.size());
mapping.links :=
    mapping.links.asSequence().select(l|l.domainMetaElement.isDefined()).sortBy(l | -
l.domainMetaElement.eAllSuperTypes.size()) +
    mapping.links.asSequence().select(l|l.domainMetaElement.isUndefined());

for (nodeMapping in GmfMap!NodeMapping.all) {
    nodeMapping.children := nodeMapping.children.asSequence().sortBy(cr|-
cr.getDomainMetaElement().eAllSuperTypes.size());
}

for (compartmentMapping in GmfMap!CompartmentMapping.all) {
    compartmentMapping.children := compartmentMapping.children.asSequence().sortBy(cr|-
cr.getDomainMetaElement().eAllSuperTypes.size());
}

-- Order tools by name

nodesToolGroup.tools := nodesToolGroup.tools.sortBy(t|t.title);
if (linksToolGroup.isDefined()) {
    linksToolGroup.tools := linksToolGroup.tools.sortBy(t|t.title);
}

-- Delete empty tool groups
if (nodesToolGroup.tools.size() = 0) delete nodesToolGroup;
if (linksToolGroup.tools.size() = 0) delete linksToolGroup;

operation ECore!EClass createLabel() {
    var labelClass := self.getLabelClass();
    var figure;

    if (labelClass.isDefined()) {
        figure := new GmfGraph!CustomFigure;
        figure.qualifiedClassName := labelClass;
    }
    else {
        figure := new GmfGraph!Label;
        figure.text := self.getLabelText();
    }

    return figure;
}

operation GmfMap!NodeReference getDomainMetaElement() {
    if (self.referencedChild.isDefined()) return self.referencedChild.domainMetaElement;
    else return self.ownedChild.domainMetaElement;
}
```

```
operation createFigureDescriptor(name : String) {
    var figureDescriptor := new GmfGraph!FigureDescriptor;
    figureDescriptor.name := name;
    figureGallery.descriptors.add(figureDescriptor);
    return figureDescriptor;
}

operation createCreationTool(element : Any) {
    var annotation : String;
    if (element.isKindOf(ECore!EClass) and element.isNode()) {
        annotation := 'gmf.node';
    }
    else {
        annotation := 'gmf.link';
    }

    var toolName := element.getAnnotationValue(annotation, 'tool.name');
    if (toolName.isUndefined()) {
        if (element.isKindOf(ECore!EClass)) {
            toolName := element.name;
        }
        else {
            toolName := element.name.firstToUpperCase();
        }
    }

    var toolDescription := element.getAnnotationValue(annotation, 'tool.description');
    if (toolDescription.isUndefined()) {
        toolDescription := 'Create new ' + toolName;
    }

    var creationTool := new GmfTool!CreationTool;
    creationTool.title := toolName;
    creationTool.description := toolDescription;

    creationTool.smallIcon := createToolImage(element.getAnnotationValue(annotation, 'tool.small.path'),
element.getAnnotationValue(annotation, 'tool.small.bundle'));
    creationTool.largeIcon := createToolImage(element.getAnnotationValue(annotation, 'tool.large.path'),
element.getAnnotationValue(annotation, 'tool.large.bundle'));

    return creationTool;
}

operation createToolImage(path : String, bundle : String) {
    if (path.isUndefined()) {
        return new GmfTool!DefaultImage;
    }
    else {
        var bundleImage := new GmfTool!BundleImage;
        bundleImage.path := path;
        bundleImage.bundle := bundle;
        return bundleImage;
    }
}

operation createReferenceCreationTool(name : String) {
    var creationTool := new GmfTool!CreationTool;
    creationTool.title := name;
    creationTool.description := 'Create new ' + name;

    creationTool.smallIcon := createRefLinkIcon();
    creationTool.largeIcon := createRefLinkIcon();
    return creationTool;
}

operation createRefLinkIcon() {
    var icon := new GmfTool!BundleImage;
    icon.bundle := 'org.eclipse.epsilon.eugenia.runtime';
    icon.path := 'icons/Link.gif';
    return icon;
}

@cached
operation getNodes() {
    return ECore!EClass.all.select(c|c.isNode());
}

@cached
operation getPhantomNodes() {
    return ECore!EClass.all.select(c|c.isPhantom());
}
```

```
@cached
operation getLinks() {
    return ECore!EClass.all.select(c|c.isLink());
}

@cached
operation getLabelledAttributesFor(class : ECore!EClass) {
    return class.eAllAttributes.select(a|a.isLabelled());
}

@cached
operation getReferenceLinks() {
    var diagramClass := getDiagramClass();
    if (diagramClass.getAnnotationValue('gmf.diagram', 'refsarelinks') = 'true') {
        return ECore!EReference.all.select(r|r.containment = false);
    }
    else {
        return ECore!EReference.all.select(r|r.isLink());
    }
}

@cached
operation ECore!EClass getAllConcreteSubTypes() {
    return ECore!EClass.all.select(c|not c.abstract and c.eAllSuperTypes.includes(self));
}

operation getDiagramClass() : ECore!EClass {
    return ECore!EClass.all.selectOne(c|c.isAnnotatedAs('gmf.diagram'));
}

operation getDiagramContainmentReference(class : ECore!EClass) {
    for (ref in getDiagramClass().getContainmentReferences()){
        if (class.eAllSuperTypes.includes(ref.eType) or class = ref.eType) return ref;
    }
}

operation getOneSuitableContainmentReference(class : ECore!EClass) {
    for (ref in ECore!EReference.all.select(sf|sf.containment)){
        if (class.eAllSuperTypes.includes(ref.eType) or class = ref.eType) return ref;
    }
}

@cached
operation getAllSuitableContainmentReferences(class : ECore!EClass) {
    var suitableReferences : Sequence;
    for (ref in ECore!EReference.all.select(sf|sf.containment)){
        if (class.eAllSuperTypes.includes(ref.eType) or class = ref.eType)
            suitableReferences.add(ref);
    }
    return suitableReferences;
}

@cached
operation ECore!EClass getContainmentReferences() {
    return self.eAllStructuralFeatures.select(sf : ECore!EReference | sf.containment);
}

@cached
operation ECore!EClass getCompartmentReferences() {
    return self.getContainmentReferences().select(r|r.isAnnotatedAs('gmf.compartment'));
}

@cached
operation ECore!EClass getAffixedReferences() {
    return self.getContainmentReferences().select(r|r.isAnnotatedAs('gmf.affixed'));
}

@cached
operation ECore!EReference isListLayout() : Boolean {
    var label := self.getAnnotationValue('gmf.compartment', 'layout');
    if (label = 'list') return true;
    else return false;
}

@cached
operation ECore!EReference isCollapsible() : Boolean {
    var label := self.getAnnotationValue('gmf.compartment', 'collapsible');
    if (label = 'false') return false;
    else return true;
}

@cached
```

```
operation ECore!EReference getLinkLabel() : String {
    var customText := self.getAnnotationValue('gmf.link', 'label.text');
    if (customText.isDefined()) {
        return customText;
    }

    return self.getAnnotationValue('gmf.link', 'label');
}

@cached
operation ECore!EReference getLongName() : String {
    return self.eContainingClass.name + self.name.firstToUpperCase();
}

@cached
operation ECore!EReference getLinkIncoming() : Boolean {
    return self.getAnnotationValue('gmf.link', 'incoming') = 'true';
}

@cached
operation ECore!EClass getConcreteSubtypes() {
    return ECore!EClass.all.select(e|(not e.abstract) and (e.eAllSuperTypes.includes(self) or e = self));
}

@cached
operation ECore!EClass getNodeSize() {
    var size := self.getAnnotationValue('gmf.node', 'size');
    if (not size.isDefined()) return size;
    else {
        var d : new GmfGraph!Dimension;
        d.dx := size.split(',').at(0).asInteger();
        d.dy := size.split(',').at(1).asInteger();
        return d;
    }

}

@cached
operation ECore!EClass getNodeSize() {
    var size := self.getAnnotationValue('gmf.node', 'size');
    if (not size.isDefined()) return size;
    else {
        var d : new GmfGraph!Dimension;
        d.dx := size.split(',').at(0).asInteger();
        d.dy := size.split(',').at(1).asInteger();
        return d;
    }

}
operation ECore!EClass getLinkEndFeature(name : String) {
    var featureName := self.getAnnotationValue('gmf.link', name);
    return self.eAllStructuralFeatures.selectOne(sf|sf.name = featureName);
}

@cached
operation ECore!EClass getLinkIncoming() : Boolean {
    return self.getAnnotationValue('gmf.link', 'incoming') = 'true';
}

@cached
operation ECore!EClass getLinkSourceFeature() {
    return self.getLinkEndFeature('source');
}

@cached
operation ECore!EClass getLinkTargetFeature() {
    return self.getLinkEndFeature('target');
}

@cached
operation ECore!EClass getSourceConstraint() {
    return self.getAnnotationValue('gmf.link', 'source.constraint');
}

@cached
operation ECore!EClass getTargetConstraint() {
    return self.getAnnotationValue('gmf.link', 'target.constraint');
}

@cached
operation ECore!EReference isLink() : Boolean {
    return self.isAnnotatedAs('gmf.link');
```

```
}

@cached
operation ECore!EClass isLink() : Boolean {

    if (self.abstract) return false;

    var isLink := self.isAnnotatedAs('gmf.link');
    var isNoLink := self.isAnnotatedAs('gmf.nolink');

    if (isNoLink) return false;
    else if (isLink) return true;
    else return self.eSuperTypes.exists(s|s.isLink());

    return isLink;
}

@cached
operation ECore!EClass isNode() : Boolean {

    if (self.isLink()) return false;
    if (self.abstract) return false;

    var isNode := self.isAnnotatedAs('gmf.node');
    var isNoNode := self.isAnnotatedAs('gmf.nonode');

    if (isNoNode) return false;
    else if (isNode) return true;
    else return self.eSuperTypes.exists(s|s.isNode());

    return isNode;
}

@cached
operation ECore!EClass getLabelName() : String {
    return self.name + 'Label';
}

@cached
operation ECore!EClass getLabelPlacement() : String {
    var labelPosition := self.getAnnotationValue('gmf.node', 'label.placement');
    if (labelPosition.isUndefined()) {
        return 'internal';
    } else {
        return labelPosition;
    }
}

@cached
operation ECore!EClass labelHasIcon(defaultValue : Boolean) : Boolean {

    var ann : String;
    if (self.isNode()) ann := 'gmf.node';
    else ann := 'gmf.link';

    var hasIcon := self.getAnnotationValue(ann, 'label.icon');

    if (hasIcon.isUndefined()) return defaultValue;
    else if (hasIcon = 'true') return true;
    else return false;

}

@cached
operation ECore!EClass getLabelViewPattern() {
    var ann : String;
    if (self.isNode()) ann := 'gmf.node';
    else ann := 'gmf.link';
    return self.getLabelViewPattern(ann);
}

@cached
operation ECore!EClass getLabelEditPattern() {
    var ann : String;
    if (self.isNode()) ann := 'gmf.node';
    else ann := 'gmf.link';
    return self.getLabelEditPattern(ann);
}

@cached
operation ECore!EClass getLabelParser() {
    var ann : String;
```

```
    if (self.isNode()) ann := 'gmf.node';
    else ann := 'gmf.link';
    return self.getLabelParser(ann);
}

@cached
operation ECore!EClass getLabelText() {
    var ann : String;
    if (self.isNode()) ann := 'gmf.node';
    else ann := 'gmf.link';

    var customText := self.getAnnotationValue(ann, 'label.text');
    if (customText.isDefined()) {
        return customText;
    }
    else {
        return self.name;
    }
}

@cached
operation ECore!EClass isPhantom() {
    if (self.isNode()) {
        return self.getAnnotationValue('gmf.node', 'phantom') = 'true';
    }
    else {
        return false;
    }
}

operation ECore!EClass getLabelClass() {
    var ann : String;
    if (self.isNode()) ann := 'gmf.node';
    else ann := 'gmf.link';

    return self.getAnnotationValue(ann, 'label.impl');
}

operation ECore!EClass getLabelAttributes() {
    var ann : String;
    if (self.isNode()) ann := 'gmf.node';
    else ann := 'gmf.link';

    return self.getLabelAttributes(ann);
}

@cached
operation ECore!EClass getLabelReadOnly() : Boolean {
    var ann : String;
    if (self.isNode()) ann := 'gmf.node';
    else ann := 'gmf.link';

    return self.getAnnotationValue(ann, 'label.readOnly') = 'true';
}

@cached
operation ECore!EAttribute isLabelled() : Boolean {
    return self.isAnnotatedAs('gmf.label');
}

@cached
operation ECore!EAttribute getLabelName() : String {
    return self.eContainingClass.name + self.name.firstToUpperCase() + 'Label';
}

@cached
operation ECore!EAttribute getReadOnly() : Boolean {
    return self.getAnnotationValue('gmf.label', 'readOnly') = 'true';
}

@cached
operation ECore!EAttribute getLabelViewPattern() {
    return self.getLabelViewPattern('gmf.label');
}

@cached
operation ECore!EAttribute getLabelEditPattern() {
    return self.getLabelEditPattern('gmf.label');
}

@cached
operation ECore!EAttribute getLabelParser() {
```

```
    return self.getLabelParser('gmf.label');
}

@cached
operation ECore!EAttribute getLabelAttributes() {
    return self.getLabelAttributes('gmf.label');
}

@cached
operation ECore!EAnnotation getDetail(key : String) : String {
    var detail := self.details.selectOne(d|d.key = key);
    if (detail.isDefined()) {return detail.value;}
    else {return null;}
}

operation ECore!EModelElement getLabelParser(ann: String) {
    return self.getAnnotationValue(ann, 'label.parser');
}

operation ECore!EModelElement getLabelEditPattern(ann: String) {
    return self.getLabelPattern(ann, 'label.edit.pattern', 'label.pattern');
}

operation ECore!EModelElement getLabelViewPattern(ann: String) {
    return self.getLabelPattern(ann, 'label.view.pattern', 'label.pattern');
}

operation ECore!EModelElement getLabelPattern(ann: String, subtype: String, fallback: String) {
    var pattern = self.getAnnotationValue(ann, subtype);
    if (pattern.isDefined()) {
        return pattern;
    } else {
        return self.getAnnotationValue(ann, fallback);
    }
}

operation ECore!EModelElement getLabelAttributes(ann: String) {
    var labelAnnotationValue := self.getAnnotationValue(ann, 'label');

    if (labelAnnotationValue.isDefined()) {
        var labels := labelAnnotationValue.split(',').collect(s|s.trim());
        return self.eAllStructuralFeatures.select(f|labels.exists(s|s = f.name));
    }
    else {
        return Sequence {};
    }
}

operation ECore!EModelElement getAnnotationValue(name : String, detail : String) : Any {
    var ann := self.eAnnotations.selectOne(a|a.source = name);
    var det;

    if (ann.isDefined()) {
        det := ann.details.selectOne(d|d.key = detail);
    }

    if (det.isDefined()) {
        return det.value;
    }
    else if (self.isTypeOf(ECore!EClass)) {
        for (s in self.eSuperTypes) {
            var sann := s.getAnnotationValue(name, detail);
            if (sann.isDefined()) {
                return sann;
            }
        }
    }
    return det;
}

@cached
operation ECore!EModelElement getAnnotation(name : String) : ECore!EAnnotation {
    var ann := self.eAnnotations.selectOne(a|a.source = name);

    if (self.isTypeOf(ECore!EClass) and ann.isUndefined()) {
        for (s in self.eSuperTypes) {
            var sann := s.getAnnotation(name);
            if (sann.isDefined()) return sann;
        }
    }

    return ann;
```

```
}

@cached
operation ECore!EModelElement isAnnotatedAs(name : String) : Boolean {
    return self.getAnnotation(name).isDefined();
}

@cached
operation ECore!EReference isLabelled() : Boolean {
    return false;
}

operation ECore!EClass createFigure() {

    var shapeName := self.getFormatOption('figure');
    var marginSize := self.getFormatOption('margin');
    if (marginSize.isUndefined()) {
     marginSize := 5;
    } else {
     marginSize := marginSize.asInteger();
    }

    var shape;
    if (shapeName = 'rectangle') {
        shape := new GmfGraph!Rectangle;
    }
    else if (shapeName = 'ellipse') {
        shape := new GmfGraph!Ellipse;
    }
    else if (shapeName = 'polygon') {
        shape := new GmfGraph!ScalablePolygon;
        var polygonX := self.getFormatOption('polygon.x').trim().split('\\s+');
        var polygonY := self.getFormatOption('polygon.y').trim().split('\\s+');
        for (i in 0.to(polygonX.size() - 1)) {
            var point := new GmfGraph!Point;
            point.x := polygonX.get(i).asInteger();
            point.y := polygonY.get(i).asInteger();
             shape.template.add(point);
        }
    }
    else if (shapeName = 'svg') {
     shape := new GmfGraph!SVGFigure;
     shape.documentURI := self.getFormatOption('svg.uri');
    }
    else if (shapeName.isUndefined() or shapeName='rounded') {
        shape := new GmfGraph!RoundedRectangle;
        shape.cornerHeight := 8;
        shape.cornerWidth := 8;
    }
    else {
        shape := new GmfGraph!CustomFigure;
        shape.qualifiedClassName := shapeName;
        marginSize := 0;
    }


    if (marginSize > 0) {
        var marginBorder := new GmfGraph!MarginBorder;
        var insets := new GmfGraph!Insets;
        var border := marginSize;
        insets.top := border;
        insets.bottom := border;
        insets.left := border;
        insets.right := border;
        marginBorder.insets := insets;
        shape.border := marginBorder;
    }

    shape.formatNode(self);

    return shape;
}

operation GmfGraph!Figure formatLine(color : String, width : String, style : String) {

    if (color.isDefined()) {
        self.foregroundColor := createColor(color);
    }

    if (width.isDefined() and self.isKindOf(GmfGraph!Shape)) {
        self.lineWidth := width.asInteger();
```

```
    }

    if (style.isDefined() and self.isKindOf(GmfGraph!Shape)) {
        if (style = 'dash') {
            self.lineKind := GmfGraph!LineKind#LINE_DASH;
        }
        else if (style = 'dot') {
            self.lineKind := GmfGraph!LineKind#LINE_DOT;
        }
        else if (style = 'solid') {
            self.lineKind := GmfGraph!LineKind#LINE_SOLID;
        }
    }

}

operation GmfGraph!Figure formatNode(e : ECore!EModelElement) {

    var backgroundColor := e.getFormatOption('color');

    if (backgroundColor.isDefined()) {
        self.backgroundColor := createColor(backgroundColor);
    }

    var size := e.getFormatOption('size');

    if (size.isDefined()) {
        self.preferredSize := createDimension(size);
    }

    self.formatLine(e.getFormatOption('border.color'),e.getFormatOption('border.width'),e.getFormatOption('border.style

}

operation GmfGraph!PolylineConnection formatConnection(e) {
    self.sourceDecoration := createPolylineDecoration(self.name + 'SourceDecoration',
e.getFormatOption('source.decoration'));
    self.targetDecoration := createPolylineDecoration(self.name + 'TargetDecoration',
e.getFormatOption('target.decoration'));

    self.formatLine(e.getFormatOption('color'),e.getFormatOption('width'),e.getFormatOption('style'));
}

operation createPolylineDecoration(name : String, type : String) {
    var polylineDecoration;

    if (type = 'none' or type.isUndefined()) {

    }
    else if (type = 'arrow') {
        polylineDecoration := new GmfGraph!PolylineDecoration;
        polylineDecoration.name := name;
    }
    else if (type = 'rhomb') {
        polylineDecoration := createRhomb(false);
    }
    else if (type = 'filledrhomb') {
        polylineDecoration := createRhomb(true);
    }
    else if (type = 'closedarrow') {
        polylineDecoration := createClosedArrow(false);
    }
    else if (type = 'filledclosedarrow') {
        polylineDecoration := createClosedArrow(true);
    }
    else if (type = 'square') {
        polylineDecoration := createSquare(false);
    }
    else if (type = 'filledsquare') {
        polylineDecoration := createSquare(true);
    }
    else {
        polylineDecoration := new GmfGraph!CustomDecoration;
        polylineDecoration.qualifiedClassName := type;
        polylineDecoration.name := name;
    }

    if (polylineDecoration.isDefined() and figureGallery.figures.excludes(polylineDecoration)) {
        figureGallery.figures.add(polylineDecoration);
    }

    return polylineDecoration;
```

```
}

@cached
operation createRhomb(filled:Boolean) : GmfGraph!PolygonDecoration {
    var rhomb := new GmfGraph!PolygonDecoration;
    rhomb.name := 'Rhomb';
    if (filled) {rhomb.name := 'Filled' + rhomb.name;}
    rhomb.template.add(createPoint(-1,1));
    rhomb.template.add(createPoint(0,0));
    rhomb.template.add(createPoint(-1,-1));
    rhomb.template.add(createPoint(-2,0));
    rhomb.template.add(createPoint(-1,1));

    if (not filled) {
        var bg := new GmfGraph!ConstantColor;
        bg.value := GmfGraph!ColorConstants#white;
        rhomb.backgroundColor := bg;
    }

    return rhomb;
}

@cached
operation createClosedArrow(filled:Boolean) : GmfGraph!PolygonDecoration {
    var arrow := new GmfGraph!PolygonDecoration;
    arrow.name := 'ClosedArrow';
    if (filled) {arrow.name := 'Filled' + arrow.name;}

    arrow.template.add(createPoint(0,0));
    arrow.template.add(createPoint(-2,2));
    arrow.template.add(createPoint(-2,-2));
    arrow.template.add(createPoint(0,0));

    if (not filled) {
        var bg := new GmfGraph!ConstantColor;
        bg.value := GmfGraph!ColorConstants#white;
        arrow.backgroundColor := bg;
    }

    return arrow;
}

@cached
operation createSquare(filled:Boolean) : GmfGraph!PolygonDecoration {
    var rect := new GmfGraph!PolygonDecoration;
    rect.name := 'Sqare';
    if (filled) {rect.name := 'Filled' + rect.name;}

    rect.template.add(createPoint(0,1));
    rect.template.add(createPoint(-1,1));
    rect.template.add(createPoint(-1,-1));
    rect.template.add(createPoint(0,-1));
    rect.template.add(createPoint(0,1));

    if (not filled) {
        var bg := new GmfGraph!ConstantColor;
        bg.value := GmfGraph!ColorConstants#white;
        rect.backgroundColor := bg;
    }

    return rect;
}

operation createPoint(x:Integer,y:Integer) : GmfGraph!Point {
    var p : new GmfGraph!Point;
    p.x := x;
    p.y := y;
    return p;
}

operation createColor(rgb : String) : GmfGraph!Color {
    var color := new GmfGraph!RGBColor;
    var parts := rgb.split(',');
    color.red := parts.at(0).asInteger();
    color.green := parts.at(1).asInteger();
    color.blue := parts.at(2).asInteger();
    return color;
}

operation createDimension(size : String) : GmfGraph!Dimension {
    var parts := size.split(',');
```

```
    var dimension := new GmfGraph!Dimension;
    dimension.dx := parts.first.asInteger();
    dimension.dy := parts.last.asInteger();
    return dimension;
}

operation ECore!EModelElement getFormatOption(option : String) : String {
    var value := self.getAnnotationValue('gmf.node', option);
    if (value.isUndefined()) value := self.getAnnotationValue('gmf.link', option);

    return value;
}
```