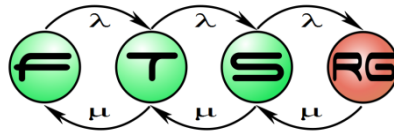


Grafikus szerkesztők fejlesztése

A Graphical Editing Framework



A GEF célja

- Grafikus szerkesztőprogramok
- Integráció az Eclipse környezetbe
- Tetszőleges modell megjelenítése
- Magas absztrakciós szint

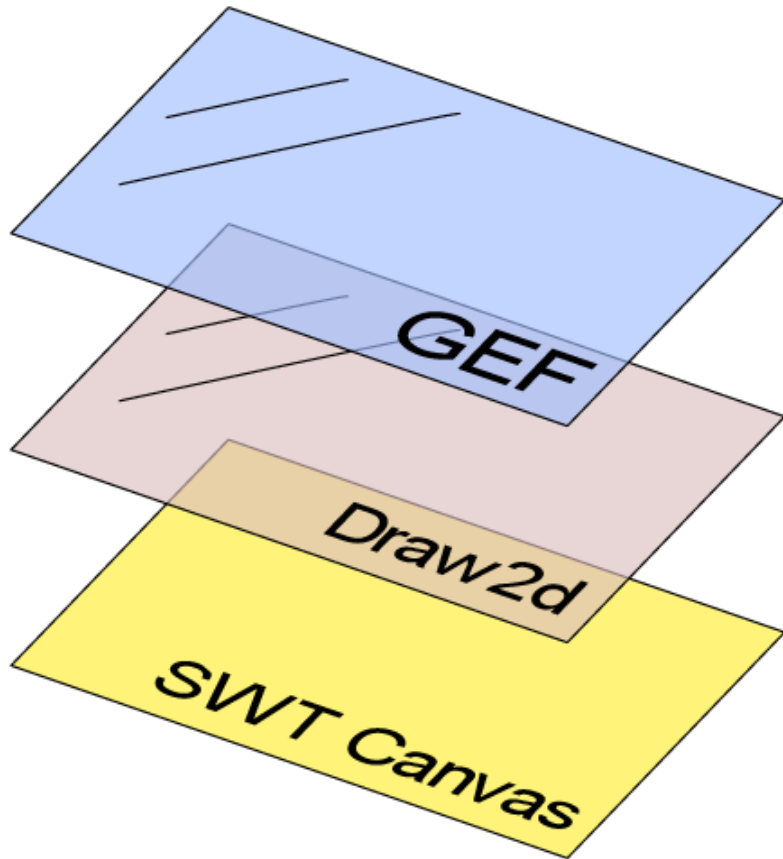
Példa


The image shows a software interface for editing Petri nets. The main window, titled 'r2init2r.vpml', displays a Petri net diagram labeled 'pn1'. The diagram consists of three places: p1 (containing 2 tokens), p2 (containing 0 tokens), and p3 (containing 0 tokens). There are two transitions: t1 and t2. Arcs connect p1 to t1, t1 to p2, p1 to t2, and t2 to p3. A toolbar on the left provides options for selecting, marqueeing, creating new places, transitions, tokens, arcs, and deleting elements or tokens. At the bottom, there are tabs for 'Petri net' and 'State machine'.

The 'Outline' window on the right shows the hierarchical structure of the Petri net model elements:

- PetriNet model elements
 - pn0
 - pn1
 - p1
 - p1_t1
 - p1_t2
 - token0
 - token1
 - p2
 - p3
 - t1
 - t1_p2
 - t2
 - t2_p3
- PetriNet diagrams
 - Example Petri net [PetriNetDiagram]
 - Example Petri net [PetriNetRoot]
 - pn1 [PetriNetFigure]
 - p1 [PlaceFigure]
 - token0 [TokenFigure]
 - token1 [TokenFigure]
 - p2 [PlaceFigure]
 - p3 [PlaceFigure]
 - t1 [TransitionFigure]
 - t2 [TransitionFigure]
 - p1_t1 [OutArcFigure]
 - p1_t2 [OutArcFigure]
 - t1_p2 [InArcFigure]
 - t2_p3 [InArcFigure]

A GEF felépítése



- Interakció (MVC)
- Modell  nézet leképezés
- Eclipse integráció

- Megjelenítés
- Elemek elrendezése
- Nagyítás

- Natív (SWT) réteg

MVC felépítés

- Model-View-Controller (Modell – Nézet – Vezérlő)
- Adatok tárolása és megjelenítése egymástól elválasztva
- Modell: adatok tárolása
- Nézet: grafikus megjelenítés
- Vezérlő: felhasználói interakció
- Elterjedt: Swing, JFace, MFC, JSF

MVC a gyakorlatban

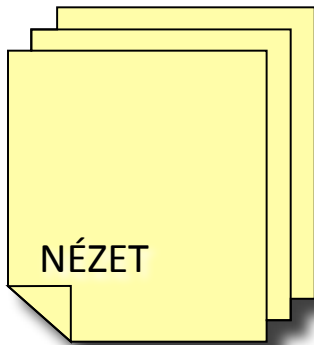
Felhasználó



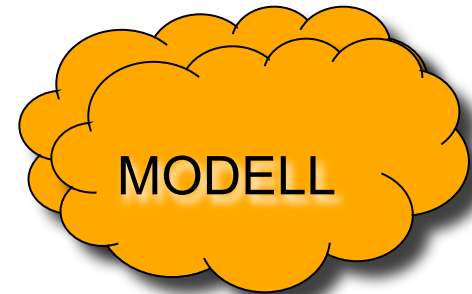
VEZÉRLŐ



NÉZET



MODELL



MVC a gyakorlatban

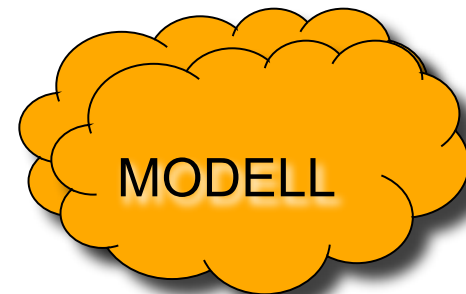
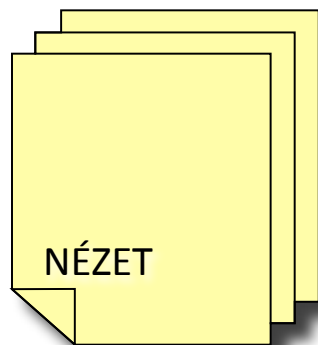
Felhasználó



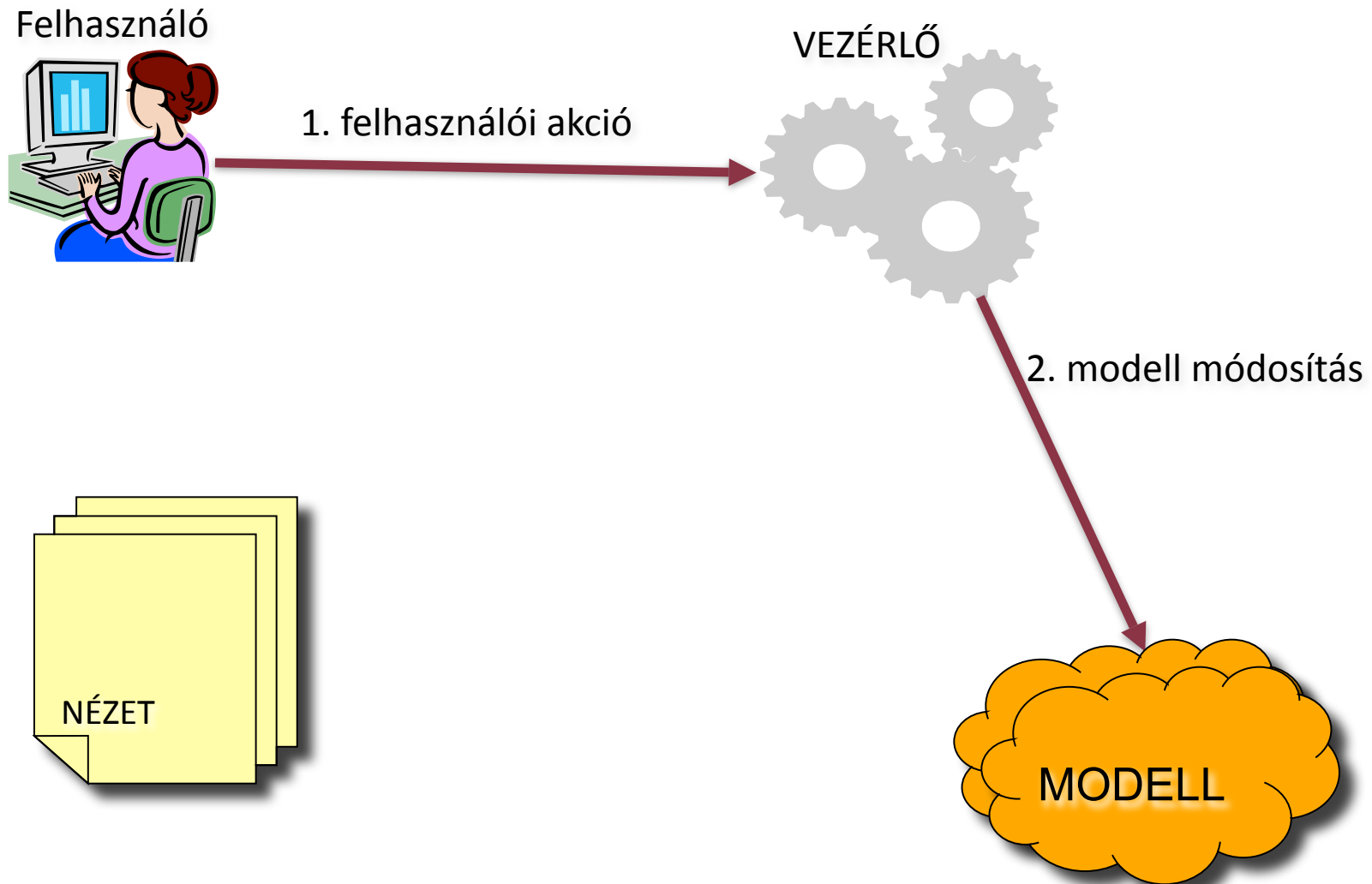
1. felhasználói akció



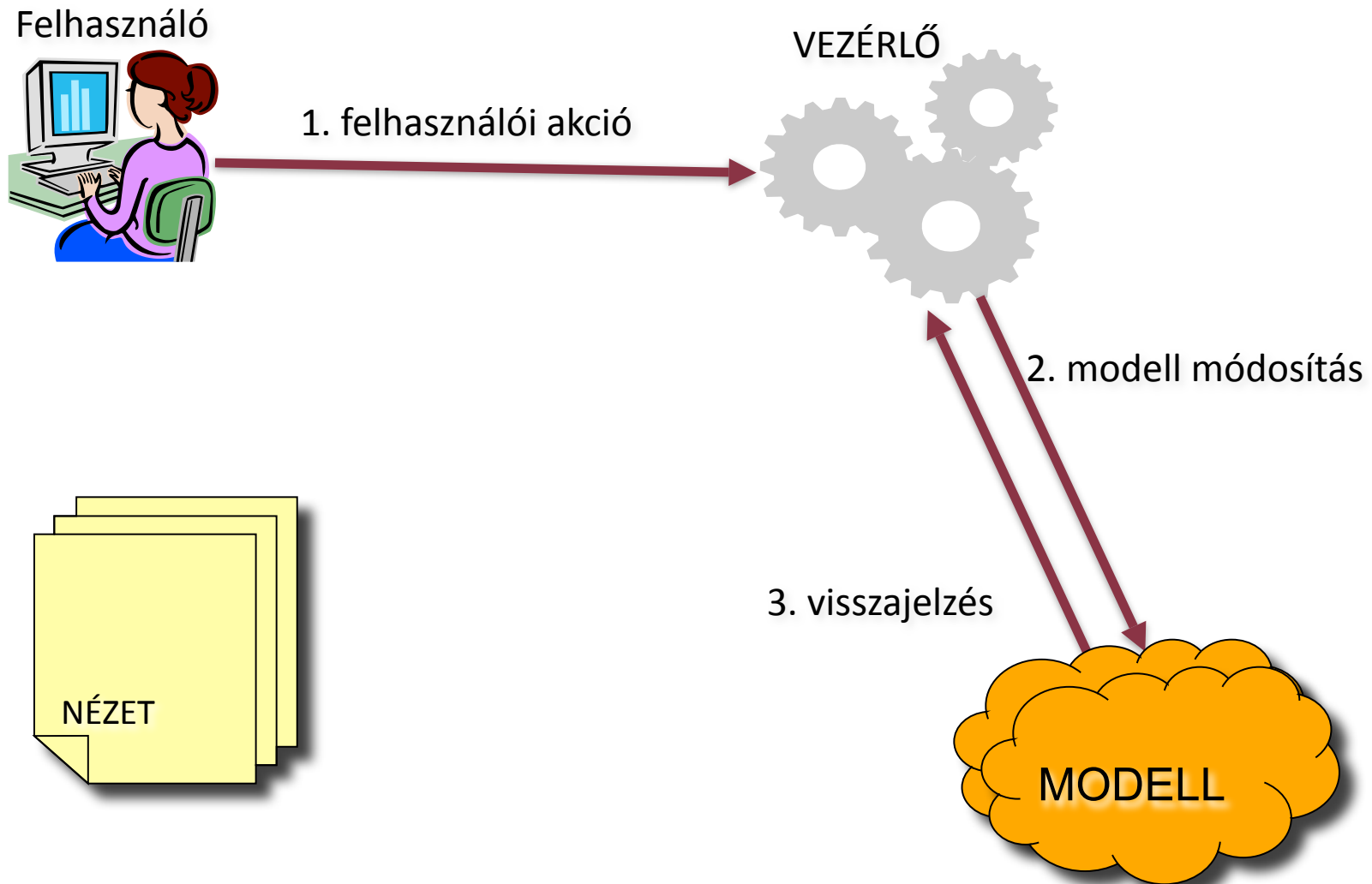
VEZÉRLŐ



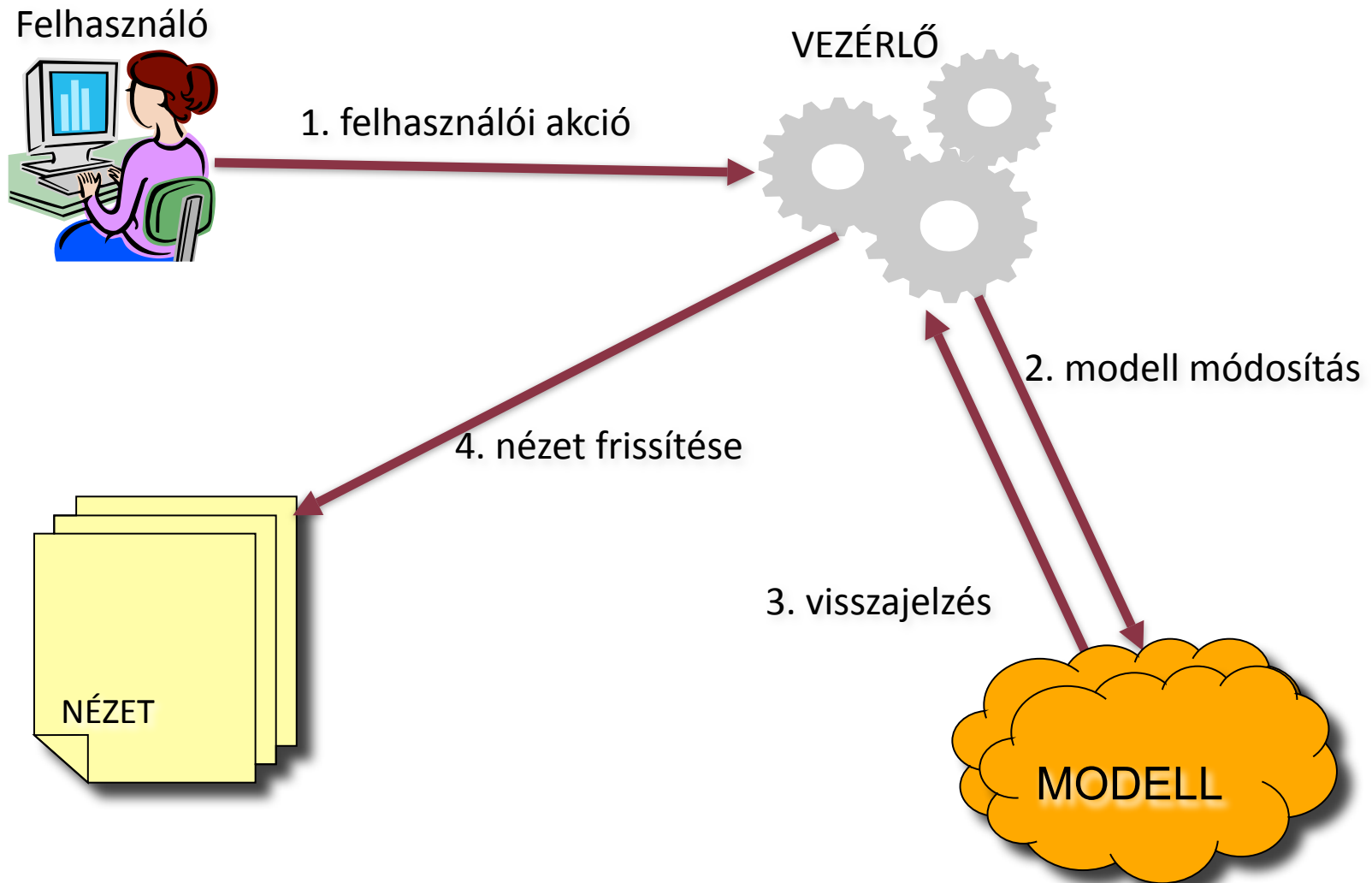
MVC a gyakorlatban



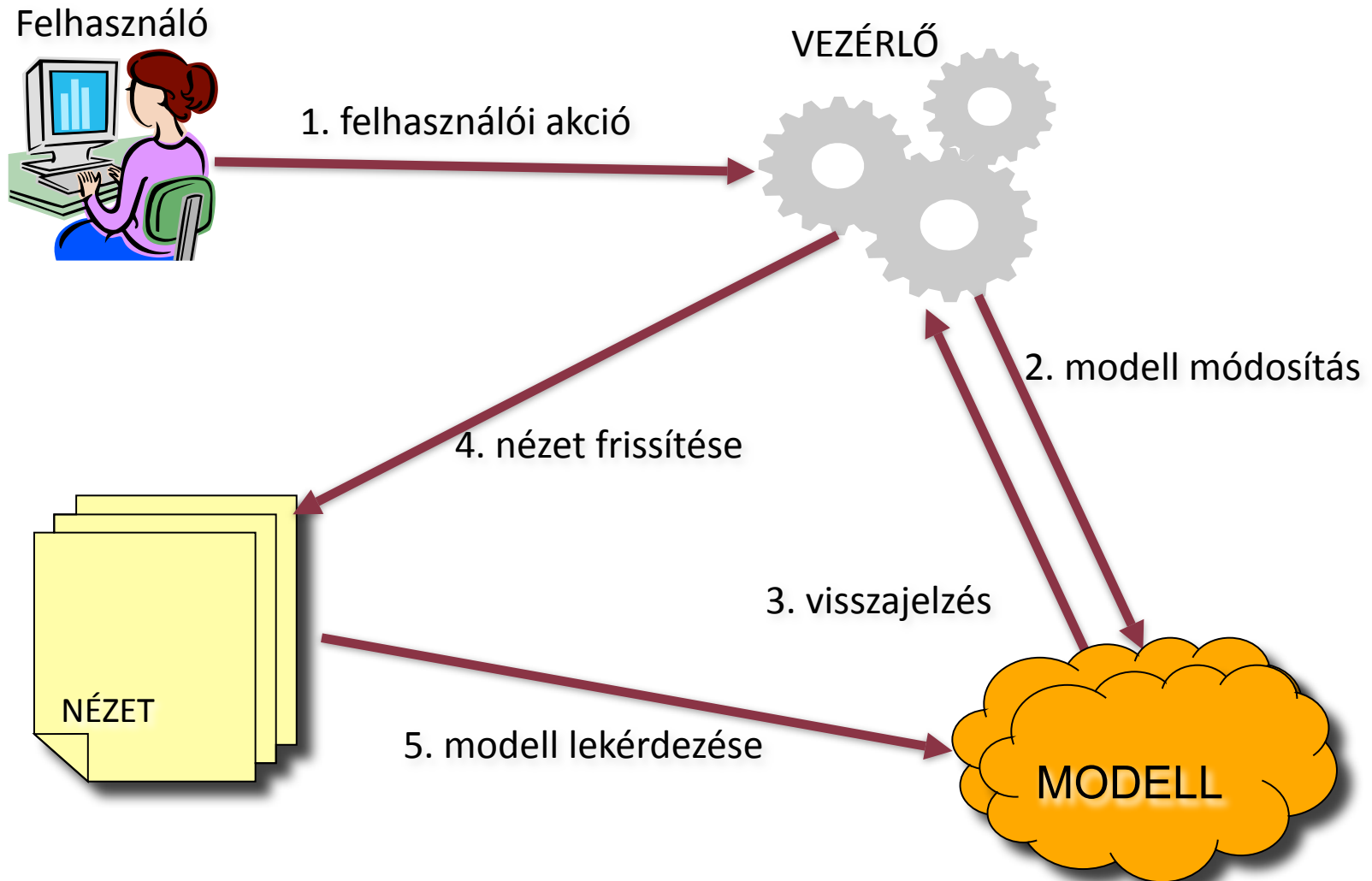
MVC a gyakorlatban



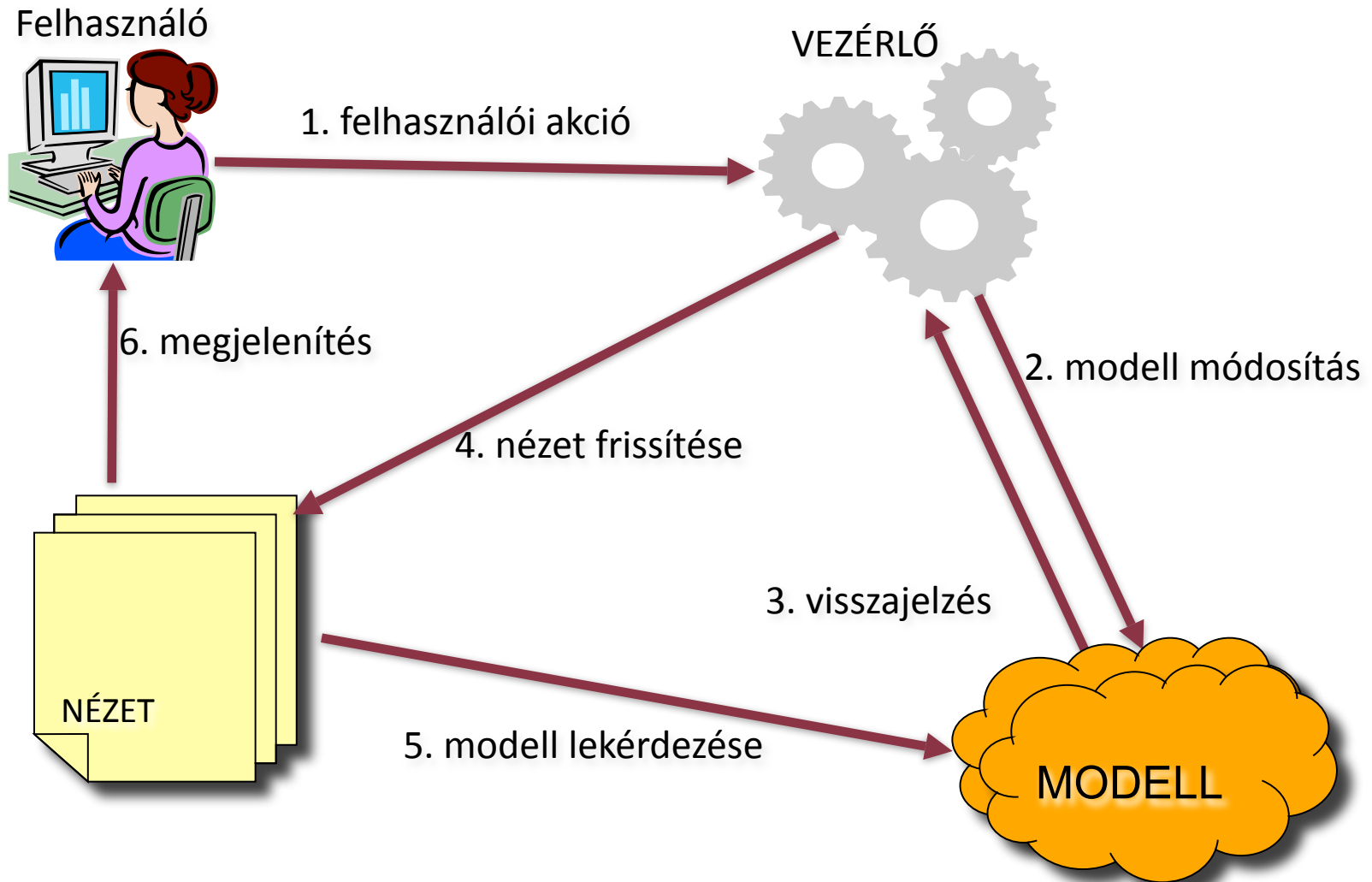
MVC a gyakorlatban



MVC a gyakorlatban



MVC a gyakorlatban



MVC a GEF-ben


Model

MVC a GEF-ben

Model

Értesítés

MVC a GEF-ben: Model

- Modell: tetszőleges
 - Pl. Java osztályok, EMF, adatbázis
 - Hierarchikus felépítés (gyökeres fa)
 - Támogatnia kell az értesítéseket
 - Jelentés a vezérlőnek, ha módosítás történt
 - 1 modell  több nézet esetén fontos
 - Pl. EMF Notification Framework
 - Üzleti modell: struktúra, adatok
 - Nézeti modell: megjelenítési információk
 - Pl. pozíció, méret

Egyszerű modell

```
public class TestModel {  
    private String name;  
    private Rectangle bounds;  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public Rectangle getBounds() {  
        return bounds;  
    }  
    public void setBounds(Rectangle bounds) {  
        this.bounds = bounds;  
    }  
}
```


Egyszerű modell

```
public class TestModel {  
    private String name;  
    private Rectangle bounds;  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public Rectangle getBounds() {  
        return bounds;  
    }  
    public void setBounds(Rectangle bounds) {  
        this.bounds = bounds;  
    }  
}
```

Belső modell

Egyszerű modell

```
public class TestModel {  
    private String name;  
    private Rectangle bounds;  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public Rectangle getBounds() {  
        return bounds;  
    }  
    public void setBounds(Rectangle bounds) {  
        this.bounds = bounds;  
    }  
}
```

Egyszerű modell

```
public class TestModel {  
    private String name;  
    private Rectangle bounds;  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public Rectangle getBounds() {  
        return bounds;  
    }  
    public void setBounds(Rectangle bounds) {  
        this.bounds = bounds;  
    }  
}
```

Nézeti modell

Egyszerű értesítés

```
public interface MyModelListener {
    public void modelChanged();
}

public class TestModel {
    ...
    private final List<MyModelListener> listeners =
        new ArrayList<MyModelListener>();
    public void addListener(MyModelListener listener) {
        listeners.add(listener);
    }
    public void removeListener(MyModelListener listener) {
        listeners.remove(listener);
    }
    protected void fireListeners() {
        for (MyModelListener listener : listeners) {
            listener.modelChanged();
        }
    }
}
```

Egyszerű értesítés

```
public interface MyModelListener {
    public void modelChanged();
}

public class TestModel {
    ...
    private final List<MyModelListener> listeners =
        new ArrayList<MyModelListener>();

    public void addListener(MyModelListener listener) {
        listeners.add(listener);
    }

    public void removeListener(MyModelListener listener) {
        listeners.remove(listener);
    }

    protected void fireListeners() {
        for (MyModelListener listener : listeners) {
            listener.modelChanged();
        }
    }
}
```

Figyelők
kezelése

Egyszerű értesítés

```
public interface MyModelListener {
    public void modelChanged();
}

public class TestModel {
    ...
    private final List<MyModelListener> listeners =
        new ArrayList<MyModelListener>();
    public void addListener(MyModelListener listener) {
        listeners.add(listener);
    }
    public void removeListener(MyModelListener listener) {
        listeners.remove(listener);
    }
    protected void fireListeners() {
        for (MyModelListener listener : listeners) {
            listener.modelChanged();
        }
    }
}
```

Egyszerű értesítés

```
public interface MyModelListener {
    public void modelChanged();
}

public class TestModel {
    ...
    private final List<MyModelListener> listeners =
        new ArrayList<MyModelListener>();
    public void addListener(MyModelListener listener) {
        listeners.add(listener);
    }
    public void removeListener(MyModelListener listener) {
        listeners.remove(listener);
    }
    protected void fireListeners() {
        for (MyModelListener listener : listeners) {
            listener.modelChanged();
        }
    }
}
```

Értesítés
küldése

Push vagy Pull értesítés

- Pull: Csak annyit küld, hogy változás történt
 - Gyors, erőforráskímélő
 - Minden változó attribútumot vizsgálni kell
- Push: Pontosán megmondjuk, hogy mi változott (pl. új X pozíció = 172)
 - El kell küldeni magát a változást is, lassú
 - Könnyen feldolgozható

GEF workflow

Model

View

GEF workflow

Model

View

Kirajzolás

GEF workflow

Model

View

Kirajzolás

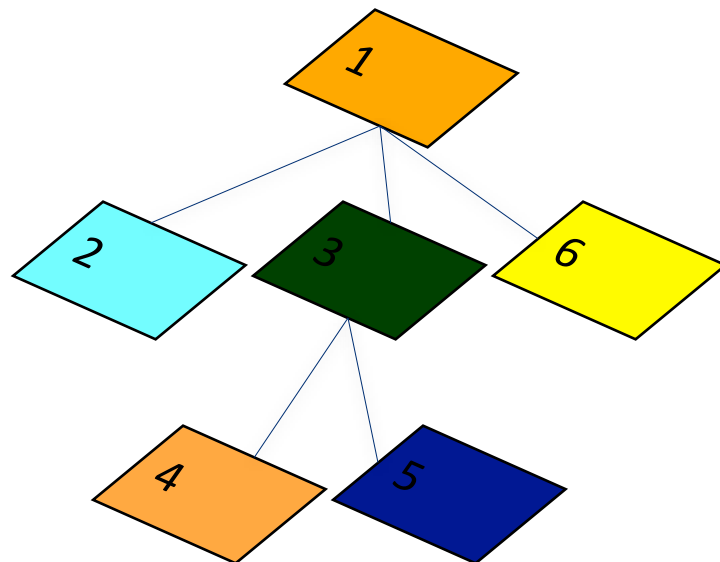
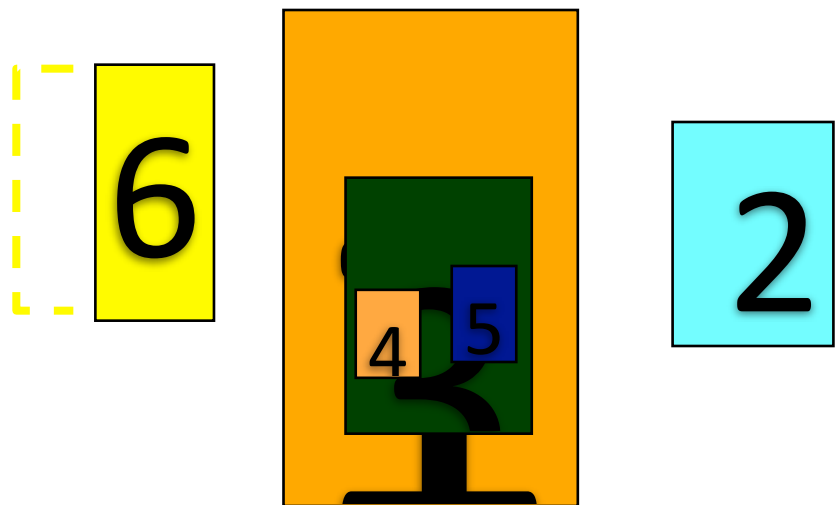
Elrendezés

MVC a GEF-ben: View

- Nézet: Draw2D osztályok
 - SWT-re épülő grafikus könyvtár
 - Egyszerű elemek (címké, téglalap, nyíl)
 - Hierarchikus megjelenítés
 - Alap építőelem: Figure
 - GEF nézet = Draw2D Figure példány
 - Bármely SWT alkalmazásban használható
 - Saját üzenetkezelése is van

Draw2D hierarchia

- Gyerek pozíciója lehet relatív a szülőhöz képest
- Gyermek negatív koordináta felé (balra, felfele) levágva
- Pontosan 1 gyökérelem

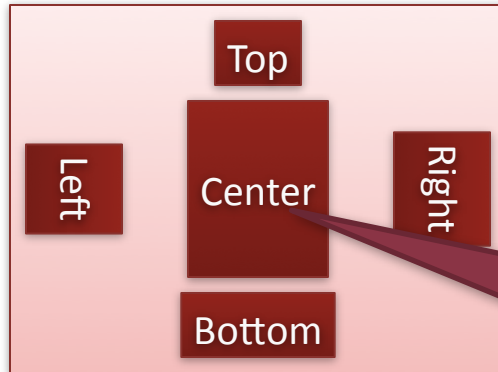


Draw2D LayoutManager

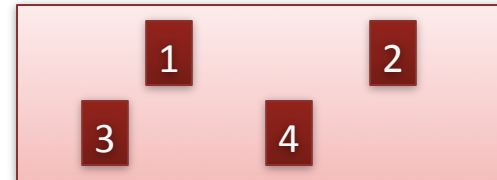
- Gyerekek elrendezése szülőn belül
- Több beépített, lehet saját is
- Constraint: Egy gyerek elhelyezésével kapcsolatos kényszer
 - SWT-nél ez volt a LayoutData
 - Szülő pozíció és méret + LayoutManager + Constraint
 - Gyerek pozíció és méret
 - A gyerekhez kell hozzárendelni
 - A szülő LayoutManagere ez alapján végzi el az elrendezést

Draw2D LayoutManagerek

BorderLayout



FlowLayout

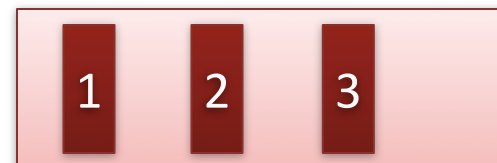


Kényszerek

XYLayout



ToolBarLayout



Draw2D alapelemek

- Egyszerű elemek
 - Label, Button, CheckBox, Image
- Geometriai alakzatok
 - RectangleFigure, Ellipse, Triangle
- Panel: általános konténer elem
- ScrollPane: görgethető tartalmú elem

Draw2D alapelemek

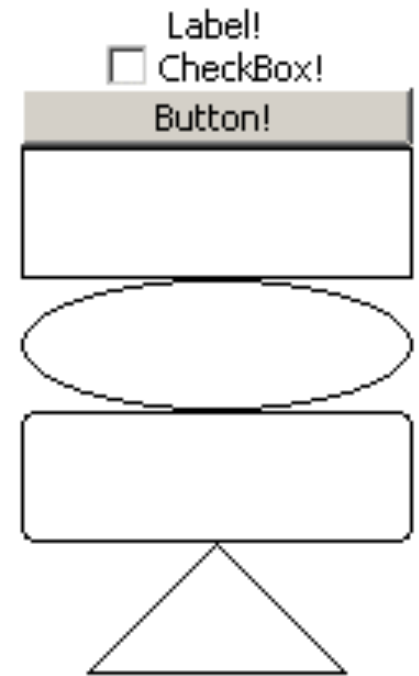
```
public class Pelda1 extends Figure {
    public Pelda1() {
        setOpaque(true);
        setBackgroundColor(ColorConstants.white);
        setLayoutManager(new ToolbarLayout());

        add(new Label("Label!"));
        add(new CheckBox("CheckBox!"));
        add(new Button("Button!"));
        add(new RectangleFigure());
        add(new Ellipse());
        add(new RoundedRectangle());
        add(new Triangle());
        for (int i = 3; i <= 6; i++)
            ((Figure) getChildren().get
                (i)).setPreferredSize(-1, 40);
    }
}
}
```

Draw2D alapelemek

```
public class Pelda1 extends Figure {
    public Pelda1() {
        setOpaque(true);
        setBackgroundColor(ColorConstants.white);
        setLayoutManager(new ToolbarLayout());

        add(new Label("Label!"));
        add(new CheckBox("CheckBox!"));
        add(new Button("Button!"));
        add(new RectangleFigure());
        add(new Ellipse());
        add(new RoundedRectangle());
        add(new Triangle());
        for (int i = 3; i <= 6; i++)
            ((Figure) getChildren().get
                (i)).setPreferredSize(-1, 40);
    }
}
```

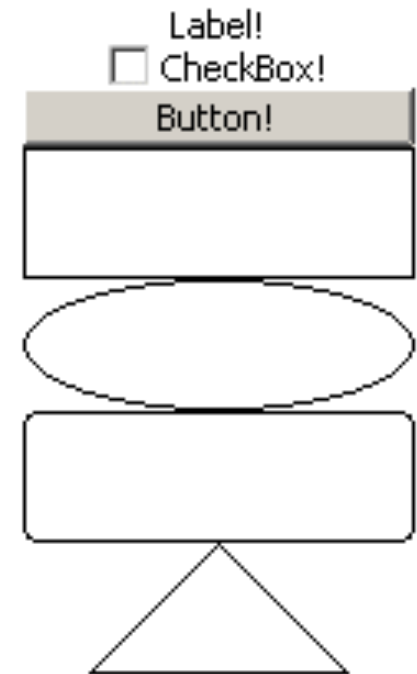


Draw2D alapelemek

```
public class Pelda1 extends JPanel {
    public Pelda1() {
        setOpaque(true);
        setBackgroundColor(ColorConstants.white);
        setLayoutManager(new ToolbarLayout());

        add(new Label("Label!"));
        add(new CheckBox("CheckBox!"));
        add(new Button("Button!"));
        add(new RectangleFigure());
        add(new Ellipse());
        add(new RoundedRectangle());
        add(new Triangle());
        for (int i = 3; i <= 6; i++)
            ((Figure) getChildren().get(i)).setPreferredSize(-1, 40);
    }
}
```

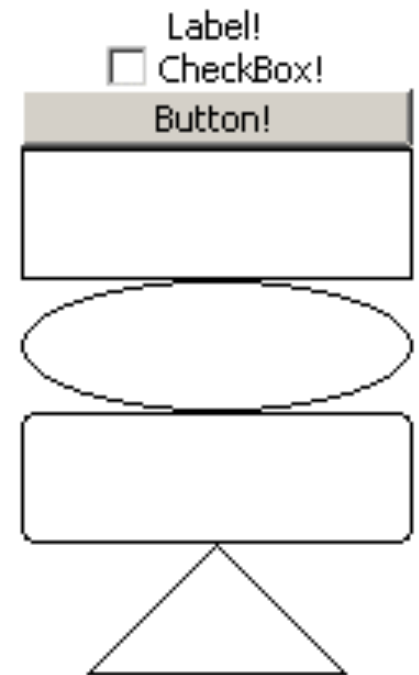
Alapértelmezetten minden átlátszó



Draw2D alapelemek

```
public class Pelda1 extends Figure {
    public Pelda1() {
        setOpaque(true);
        setBackgroundColor(ColorConstants.white);
        setLayoutManager(new ToolbarLayout());

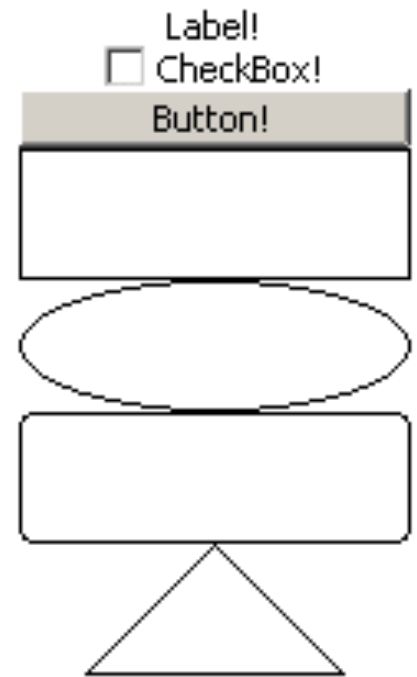
        add(new Label("Label!"));
        add(new CheckBox("CheckBox!"));
        add(new Button("Button!"));
        add(new RectangleFigure());
        add(new Ellipse());
        add(new RoundedRectangle());
        add(new Triangle());
        for (int i = 3; i <= 6; i++)
            ((Figure) getChildren().get
                (i)).setPreferredSize(-1, 40);
    }
}
```



Draw2D alapelemek

```
public class Pelda1 extends Figure {  
    public Pelda1() {  
        setOpaque(true);  
        setBackgroundColor(ColorConstants.white);  
        setLayoutManager(new ToolbarLayout());  
  
        add(new Label("Label!"));  
        add(new CheckBox("CheckBox!"));  
        add(new Button("Button!"));  
        add(new RectangleFigure());  
        add(new Ellipse());  
        add(new RoundedRectangle());  
        add(new Triangle());  
        for (int i = 3; i <= 6; i++)  
            ((Figure) getChildren().get  
                (i)).setPreferredSize(-1, 40);  
    }  
}
```

Preferált méret:
LayoutManager figyelheti



Draw2D keretek

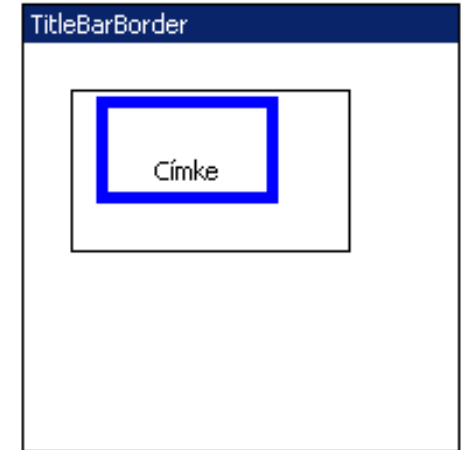
- Minden Figure-höz rendelhető keret
- TitleBarBorder: dialógusablak jelleg
- LineBorder: egyszerű vonal
- MarginBorder: üres hely
- Keretek egymásba ágyazhatók
 - CompoundBorder
- Megosztható Figure-ök között

Draw2D keretek

```
public class Example2 extends Figure {
    public Example2() {
        setOpaque(true);
        setBackgroundColor(ColorConstants.white);
        setLayoutManager(new XYLayout());
        TitleBarBorder tb = new TitleBarBorder("TitleBarBorder");
        tb.setTextColor(ColorConstants.white);
        setBorder(new CompoundBorder(new LineBorder(1), tb));
        Label lbl = new Label("Címke");
        lbl.setBorder(new CompoundBorder(
            new LineBorder(1), new CompoundBorder(
                new MarginBorder(2, 10, 20, 30),
                new LineBorder(ColorConstants.blue, 5))));
        add(lbl);
        setConstraint(lbl, new Rectangle(20, 20, 120, 70));
    }
}
```

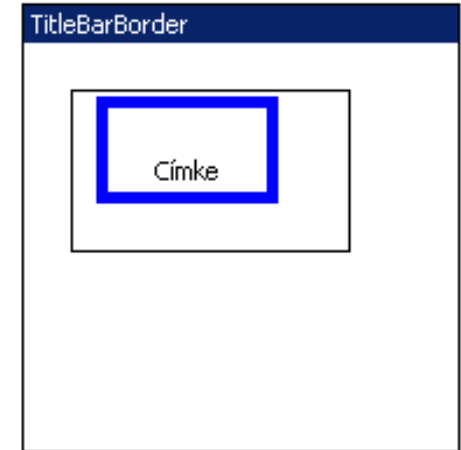
Draw2D keretek

```
public class Example2 extends Figure {
    public Example2() {
        setOpaque(true);
        setBackgroundColor(ColorConstants.white);
        setLayoutManager(new XYLayout());
        TitleBarBorder tb = new TitleBarBorder("Tit
        tb.setTextColor(ColorConstants.white);
        setBorder(new CompoundBorder(new LineBorder
        Label lbl = new Label("Címke");
        lbl.setBorder(new CompoundBorder(
            new LineBorder(1), new CompoundBorder(
            new MarginBorder(2, 10, 20, 30),
            new LineBorder(ColorConstants.blue, 5))));
        add(lbl);
        setConstraint(lbl, new Rectangle(20, 20, 120, 70));
    }
}
```



Draw2D keretek

```
public class Example2 extends Figure {
    public Example2() {
        setOpaque(true);
        setBackgroundColor(ColorConstants.white);
        setLayoutManager(new XYLayout());
        TitleBarBorder tb = new TitleBarBorder("Tit
        tb.setTextColor(ColorConstants.white);
        setBorder(new CompoundBorder(new LineBorder
        Label lbl = new Label("Címke");
        lbl.setBorder(new CompoundBorder(
            new LineBorder(1), new CompoundBorder(
            new MarginBorder(2, 10, 20, 30),
            new LineBorder(ColorConstants.blue, 5))););
        add(lbl);
        setConstraint(lbl, new Rectangle(20, 20, 120, 70));
    }
}
```



Kényszereket a
szülőnél adjuk meg

Draw2D egyéb elemek

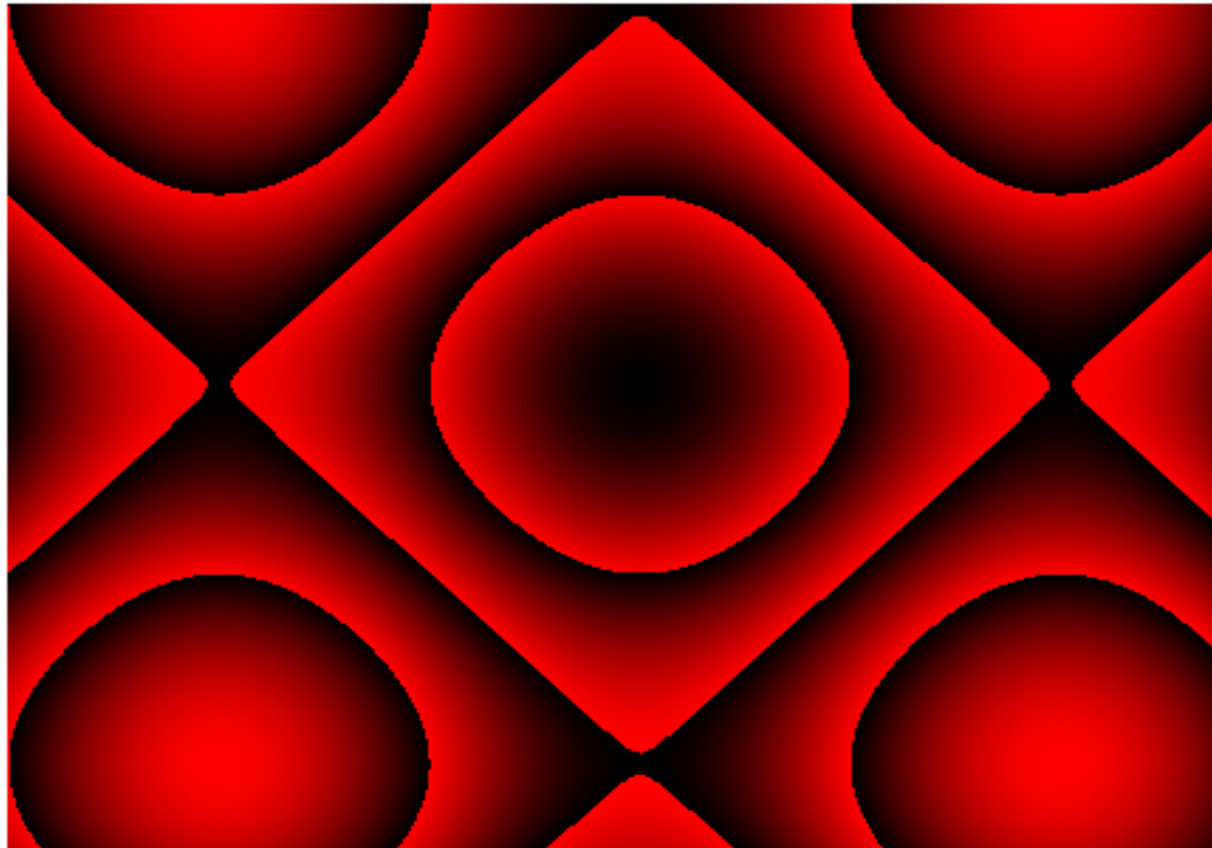
- Nyilak: lásd később
- Saját elemek
 - Beépített elemek kombinációja nem mindig elégséges
 - `paintFigure()` felülírása kell
 - Tetszőleges SWT rajzoló kód lehet

Draw2D saját elem

```
public class Example3 extends Figure {
    @Override
    protected void paintFigure(Graphics graphics) {
        Rectangle r = getBounds();
        PaletteData pd = new PaletteData(0xff0000, 0xff00, 0xff);
        pd.redShift = -16; pd.greenShift = -8; pd.blueShift = 0;
        pd.isDirect = true;
        ImageData id = new ImageData(r.width, r.height, 24, pd);
        for (int u = 0; u < r.width; u++) {
            for (int v = 0; v < r.height; v++) {
                int rc = ((int) ((Math.sin(u * 9.0 / r.width) +
                    Math.cos(v * 7.0 / r.height)) * 256.0)) % 256;
                id.setPixel(u, v, rc << 16);
            }
        }
        Image img = new Image(Display.getCurrent(), id);
        graphics.drawImage(img, r.getTopLeft());
    }
}
```

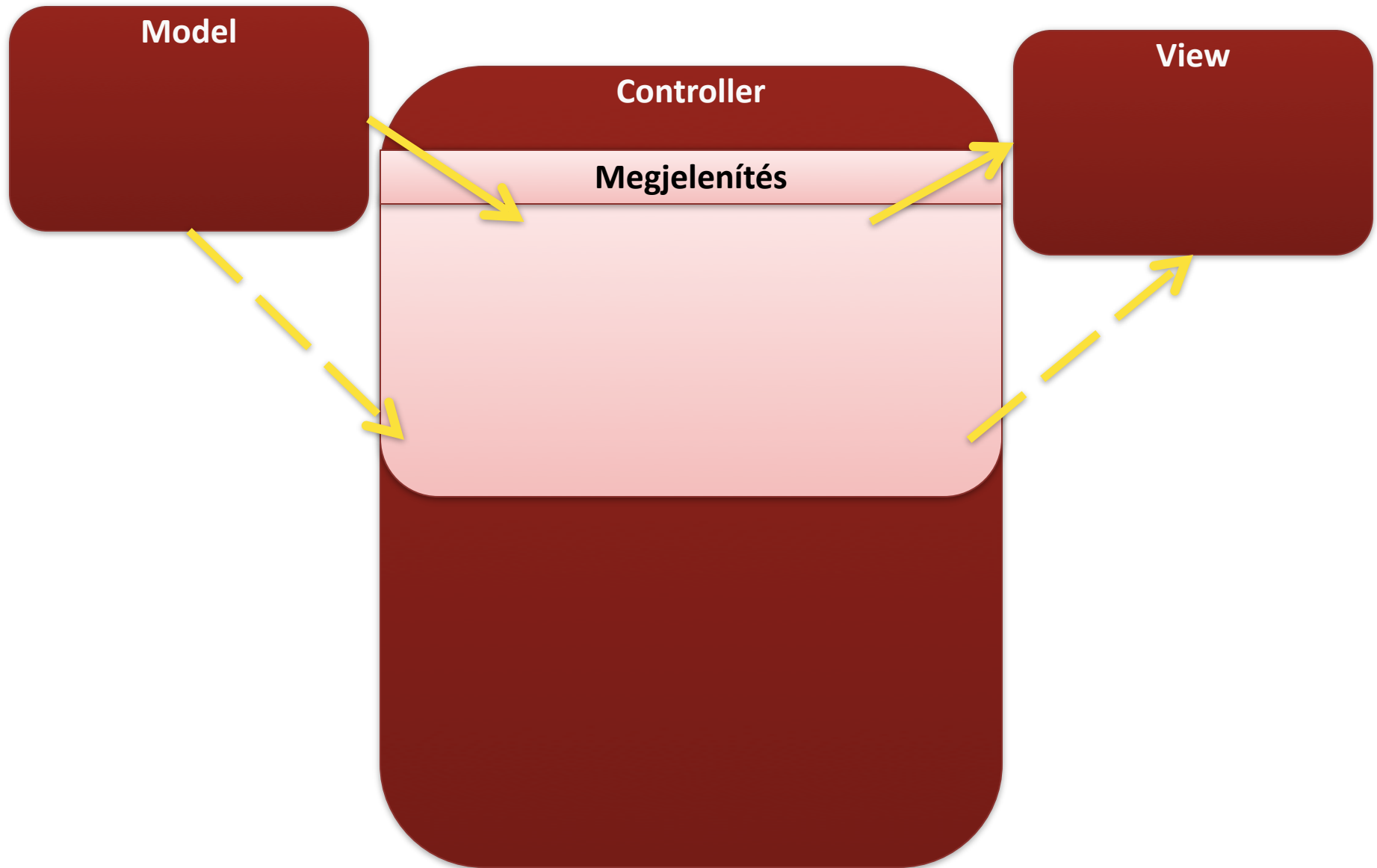
Draw2D saját elem

```
public class F...
    @Override
    protected void
        Rectang
        Palette
        pd.redS
        pd.isDi
        ImageDa
        for (ir
            for
        }
    }
    Image i
    graphic
}
}
```

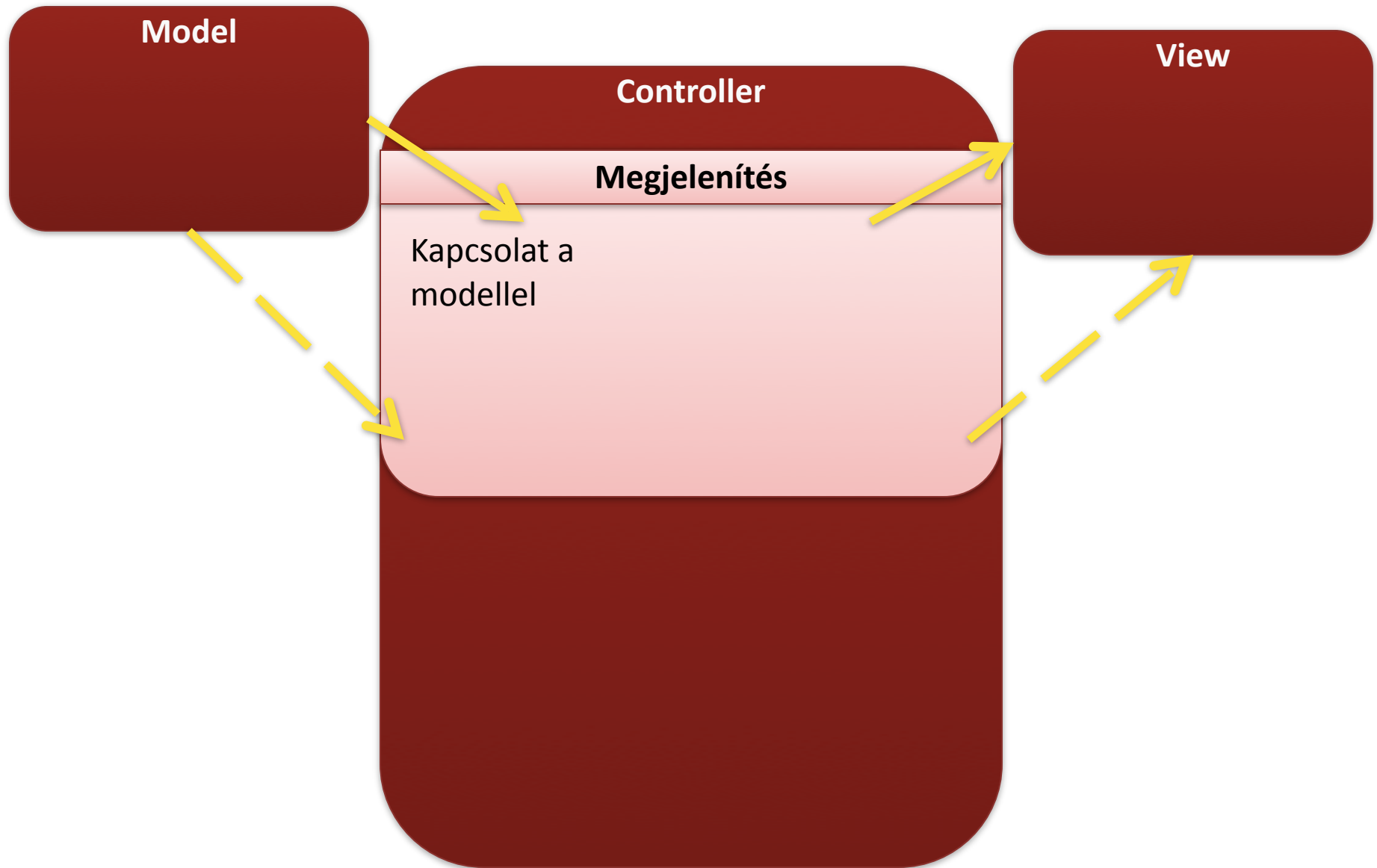


```
f);
0;
);
56;
```

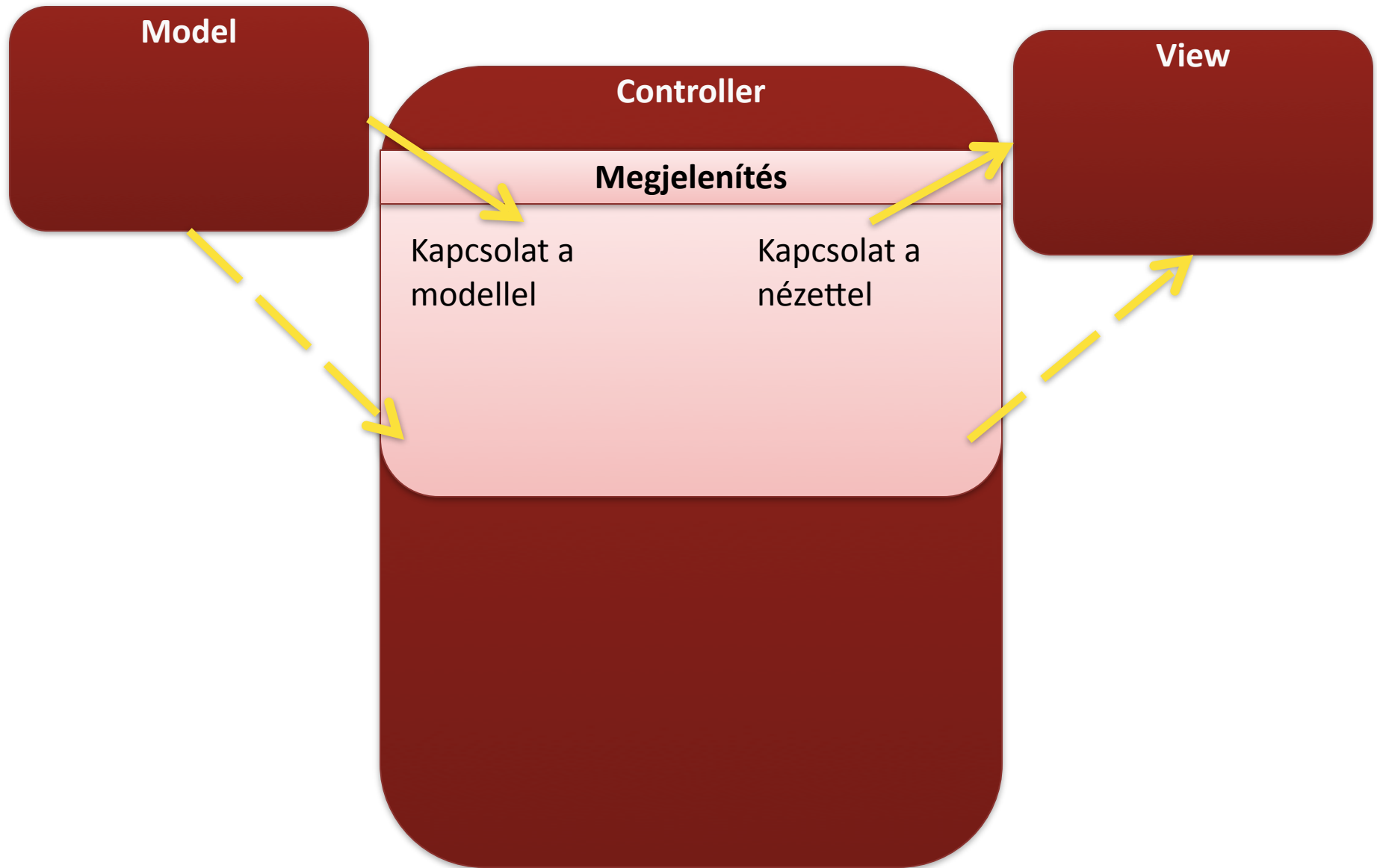
GEF workflow



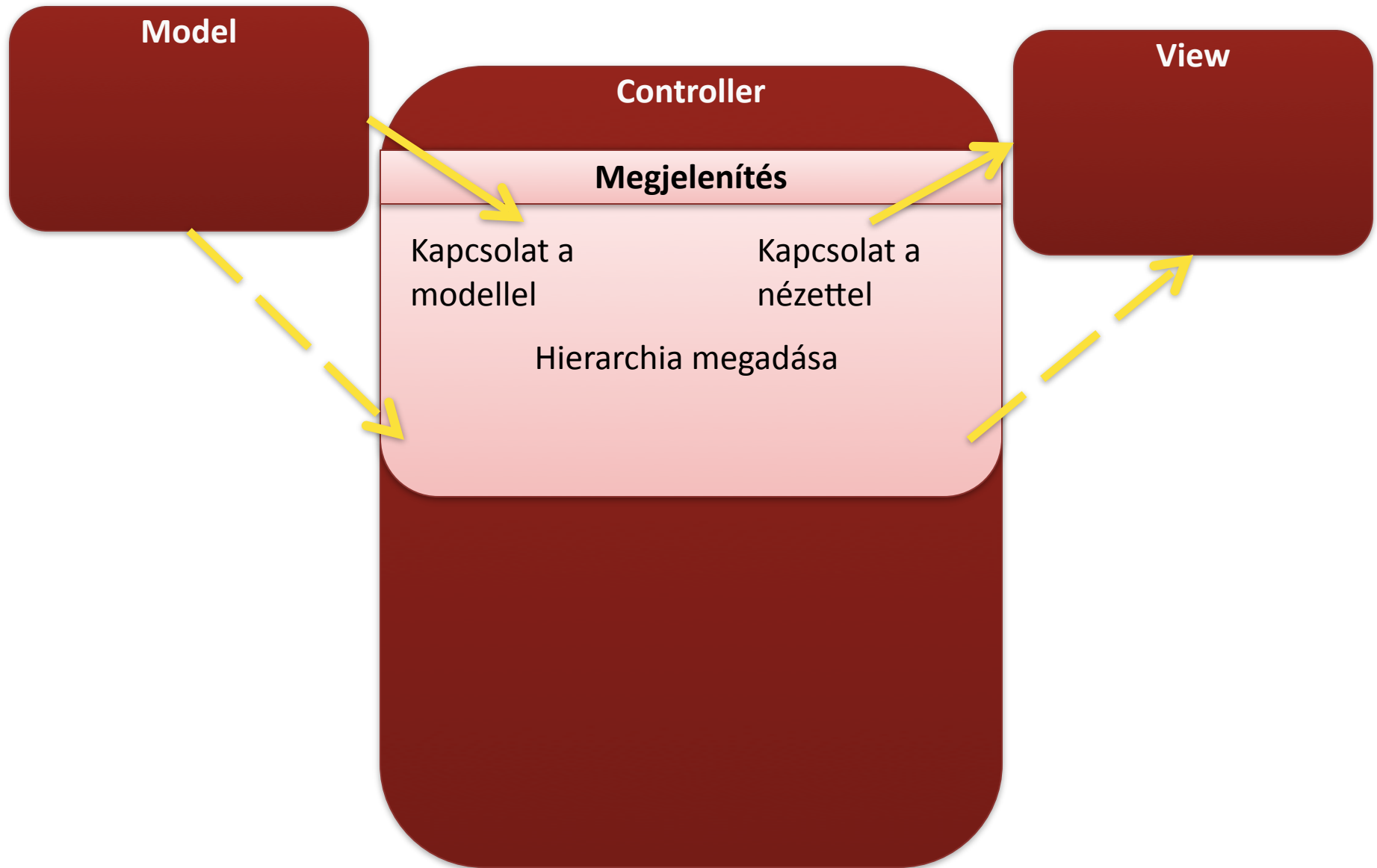
GEF workflow



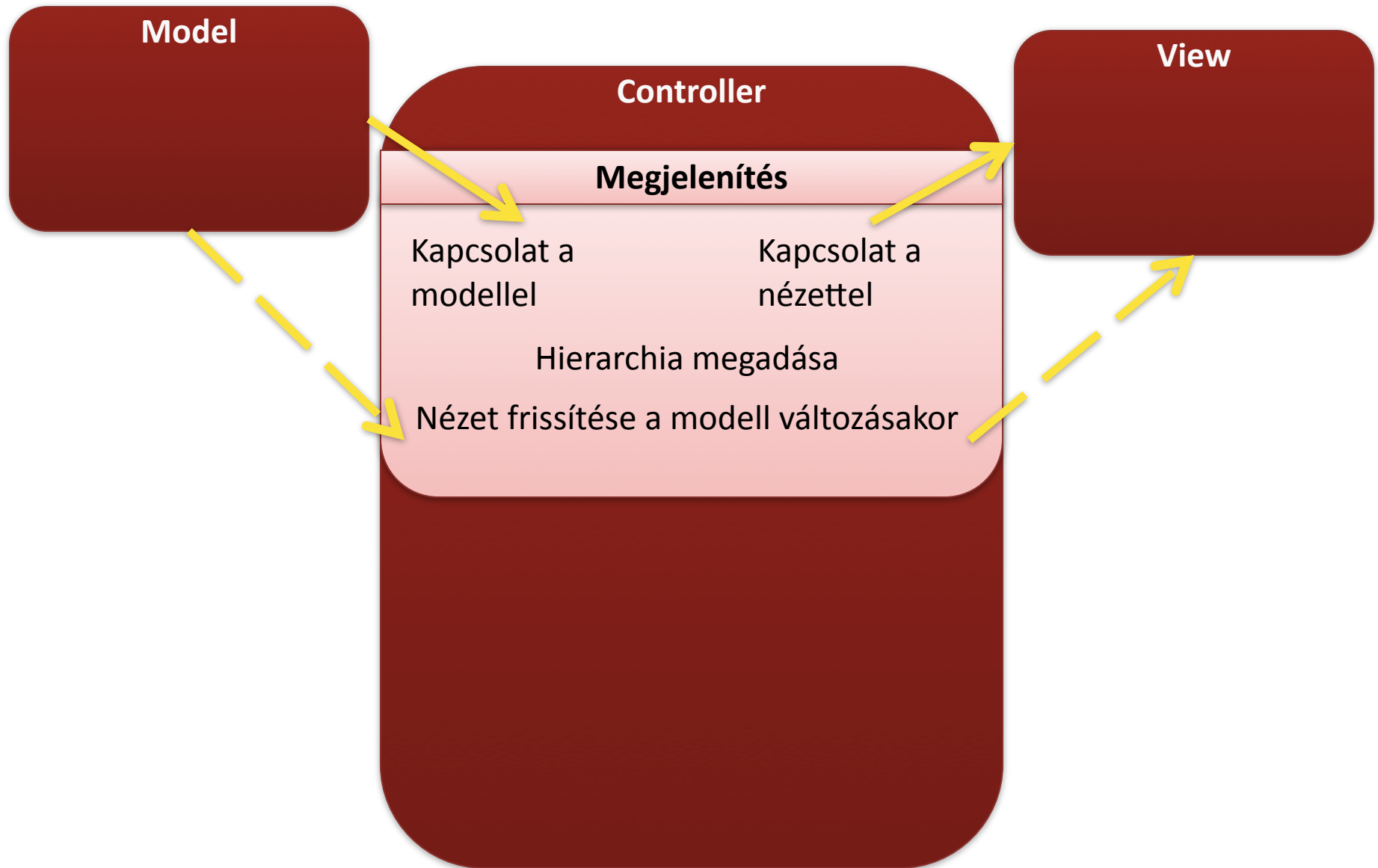
GEF workflow



GEF workflow



GEF workflow

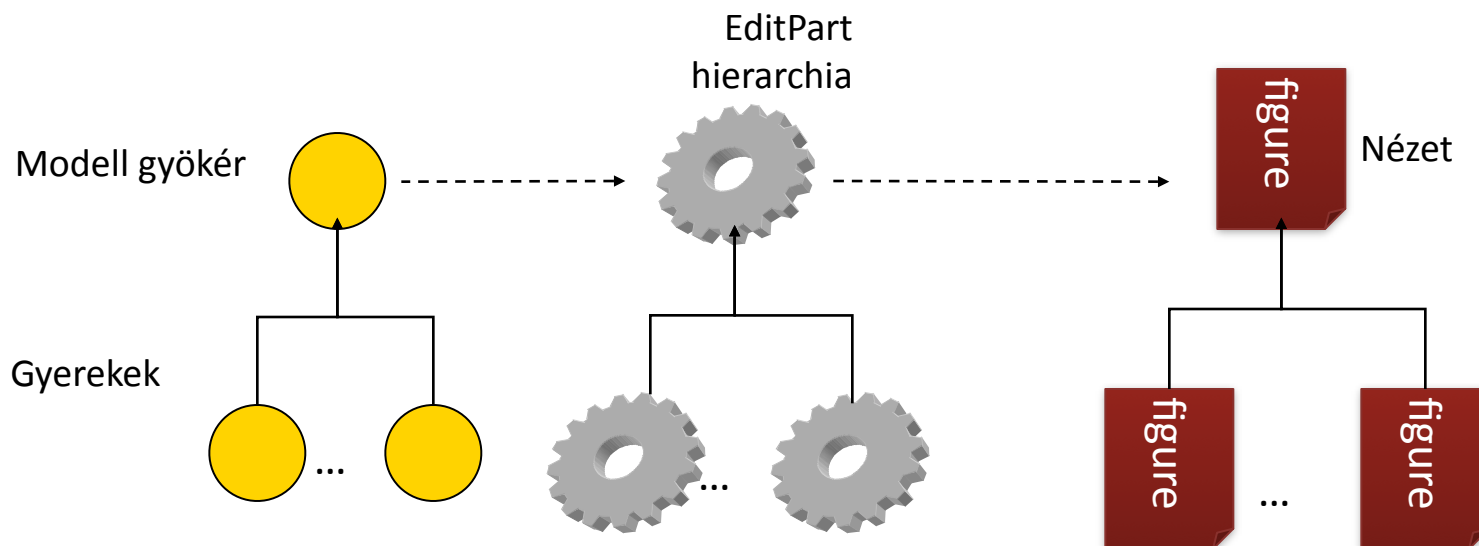


MVC a GEF-ben: Controller

- Vezérlő: EditPart osztályok
 - GEF „lelke”
 - Kapcsolat a modell és a nézet között
 - 1 Figure \leftrightarrow 1 EditPart
 - 1 modell elem \rightarrow több EditPart is lehet
 - Nézet frissítése modell alapján
 - Felhasználói akciók kezelése
 - Modell módosítása ezek alapján

Kezdeti nézet felépítése

- Modell alapján EditPartok létrehozása
 - EditPartFactory
- Nézet Figure-ök példányosítása
 - GraphicalEditPart.createFigure()



EditPartFactory

- Modell alapján EditPart generálása

```
public class TestGEFEditPartFactory implements
    EditPartFactory {
    public EditPart createEditPart (EditPart context,
    Object model) {
        EditPart ep = null;

        if (model instanceof ElementModel)
            ep = new ElementEditPart();
        else if (model instanceof ParentModel)
            ep = new ParentEditPart();

        if (ep != null)
            ep.setModel(model);
        return ep;
    }
}
```

EditPartFactory

- Modell alapján EditPart generálása

```
public class TestGEFEditPartFactory implements
    EditPartFactory {
    public EditPart createEditPart(EditPart context,
        Object model) {
        EditPart ep = null;

        if (model instanceof ElementModel)
            ep = new ElementEditPart();
        else if (model instanceof ParentModel)
            ep = new ParentEditPart();

        if (ep != null)
            ep.setModel(model);
        return ep;
    }
}
```

Szülő EditPart

EditPartFactory

- Modell alapján EditPart generálása

```
public class TestGEFEditPartFactory implements
    EditPartFactory {
    public EditPart createEditPart (EditPart context,
    Object model) {
        EditPart ep = null;

        if (model instanceof ElementModel)
            ep = new ElementEditPart();
        else if (model instanceof ParentModel)
            ep = new ParentEditPart();

        if (ep != null)
            ep.setModel(model);
        return ep;
    }
}
```

EditPartFactory

- Modell alapján EditPart generálása

```
public class TestGEFEditPartFactory implements
    EditPartFactory {
    public EditPart createEditPart(EditPart context,
        Object model) {
        EditPart ep = null;

        if (model instanceof ElementModel)
            ep = new ElementEditPart();
        else if (model instanceof ParentModel)
            ep = new ParentEditPart();

        if (ep != null)
            ep.setModel(model);
        return ep;
    }
}
```

EditPart tárol egy
modell referenciát

Nézet legenerálása

- EditPart feladata
- Sajátunkat célszerű származtatni az AbstractGraphicalEditPart osztályból

```
public class ElementEditPart extends
    AbstractGraphicalEditPart {
    ...
    @Override
    protected IFigure createFigure() {
        // Saját Figure létrehozása
        ElementFigure fig = new
ElementFigure();
        return fig;
    }
    ...
}
```


Modell bejárása

- EditPartViewer tartalma
 - EditPartFactory
 - Modell gyökér eleme
- Hogy jutunk el a modell többi részéhez?
 - EditParton keresztül
 - Mindenki megmondja a saját gyerekeit
 - Rekurzívan bejárható az egész modell
 - Ne legyen benne tartalmazás kör

Modell bejárása

- `EditPart.getModelChildren()`
 - Az `EditPart`hoz tartozó modellelem gyerekeit kell visszaadni listaként
 - Lista sorrendje számít -> nézetek takarása

```
public class TestParentEditPart extends
    AbstractGraphicalEditPart {
    ...
    @Override
    protected List getModelChildren() {
        // Saját modell lekérdezése
        ParentModel pm = ((ParentModel)
        getModel());
        return pm.getChildren();
    }
    ...
}
```

Modellfüggő, nem
GEF-specifikus

Modell bejárása

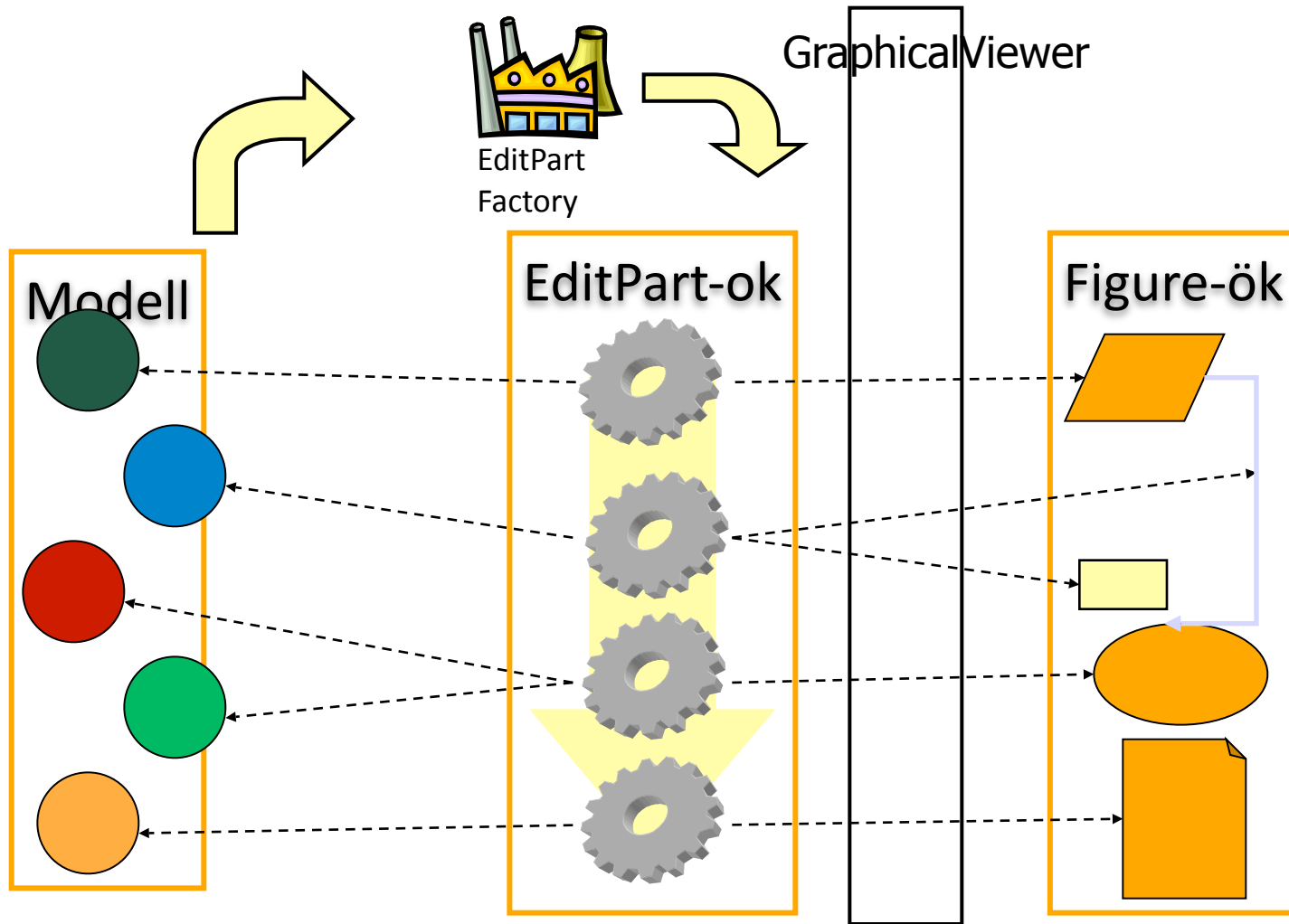
- `EditPart.getModelChildren()`
 - Az `EditPart`hoz tartozó modellelem gyerekeit kell visszaadni listaként
 - Lista sorrendje számít -> nézetek takarása

```
public class TestParentEditPart extends
    AbstractGraphicalEditPart {
    ...
    @Override
    protected List getModelChildren() {
        // Saját modell lekérése
        ParentModel pm = ((ParentModel)
        getModel());
        return pm.getChildren();
    }
    ...
}
```

Modell
lekérése

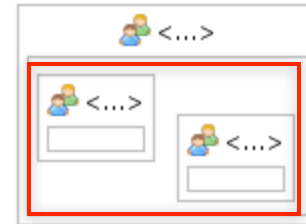
Modellfüggő, nem
GEF-specifikus

Nézet felépítés - összefoglalás



ContentPane

- ContentPane: a gyerekekhez tartozó nézeteket tartalmazó nézet
- Felülírjuk, ha egy összetett Figure-nek csak egy része tartalmazza a gyerek-Figure-öket
- EditPartban írhatjuk felül (alapesetben maga a Figure)



- ...

- @Override

```
public IFigure getContentPane() {  
    return ((MyFigure) getFigure  
        ()).getPlaceOfChildren();  
}
```

- ...

Nézet frissítése modellváltozáskor

- Modell figyelése
 - activate(), deactivate()
- Nézet frissítése
 - refreshVisuals(): nem strukturális módosítás
 - refreshChildren(): gyerekek listája változik

EditPart példa

```
public class ParentEditPart extends AbstractGraphicalEditPart
    implements MyModelListener {
    protected void refreshVisuals() {
        ((ParentView) getFigure()).setLabel(
            ((ParentModel) getModel()).getName());
    }
    public void activate() {
        super.activate();
        ((ParentModel) getModel()).addListener(this);
    }
    public void deactivate() {
        ((ParentModel) getModel()).removeListener(this);
        super.deactivate();
    }
    public void modelChanged() {
        refreshVisuals();
        refreshChildren();
    }
}
```

EditPart példa

```
public class ParentEditPart extends AbstractGraphicalEditPart
    implements MyModelListener {
    protected void refreshVisuals() {
        ((ParentView) getFigure()).setLabel(
            ((ParentModel) getModel()).getName());
    }
    public void activate() {
        super.activate();
        ((ParentModel) getModel()).addListener(this);
    }
    public void deactivate() {
        ((ParentModel) getModel()).removeListener(this);
        super.deactivate();
    }
    public void modelChanged() {
        refreshVisuals();
        refreshChildren();
    }
}
```

Nézet frissítése

EditPart példa

```
public class ParentEditPart extends AbstractGraphicalEditPart
    implements MyModelListener {
    protected void refreshVisuals() {
        ((ParentView) getFigure()).setLabel(
            ((ParentModel) getModel()).getName());
    }
    public void activate() {
        super.activate();
        ((ParentModel) getModel()).addListener(this);
    }
    public void deactivate() {
        ((ParentModel) getModel()).removeListener(this);
        super.deactivate();
    }
    public void modelChanged() {
        refreshVisuals();
        refreshChildren();
    }
}
```

Modellfigyelés
kezdeté

EditPart példa

```
public class ParentEditPart extends AbstractGraphicalEditPart
    implements MyModelListener {
    protected void refreshVisuals() {
        ((ParentView) getFigure()).setLabel(
            ((ParentModel) getModel()).getName());
    }
    public void activate() {
        super.activate();
        ((ParentModel) getModel()).addListener(this);
    }
    public void deactivate() {
        ((ParentModel) getModel()).removeListener(this);
        super.deactivate();
    }
    public void modelChanged() {
        refreshVisuals();
        refreshChildren();
    }
}
```

Modellfigyelés
vége

EditPart példa

```
public class ParentEditPart extends AbstractGraphicalEditPart
    implements MyModelListener {
    protected void refreshVisuals() {
        ((ParentView) getFigure()).setLabel(
            ((ParentModel) getModel()).getName());
    }
    public void activate() {
        super.activate();
        ((ParentModel) getModel()).addListener(this);
    }
    public void deactivate() {
        ((ParentModel) getModel()).removeListener(this);
        super.deactivate();
    }
    public void modelChanged() {
        refreshVisuals();
        refreshChildren();
    }
}
```

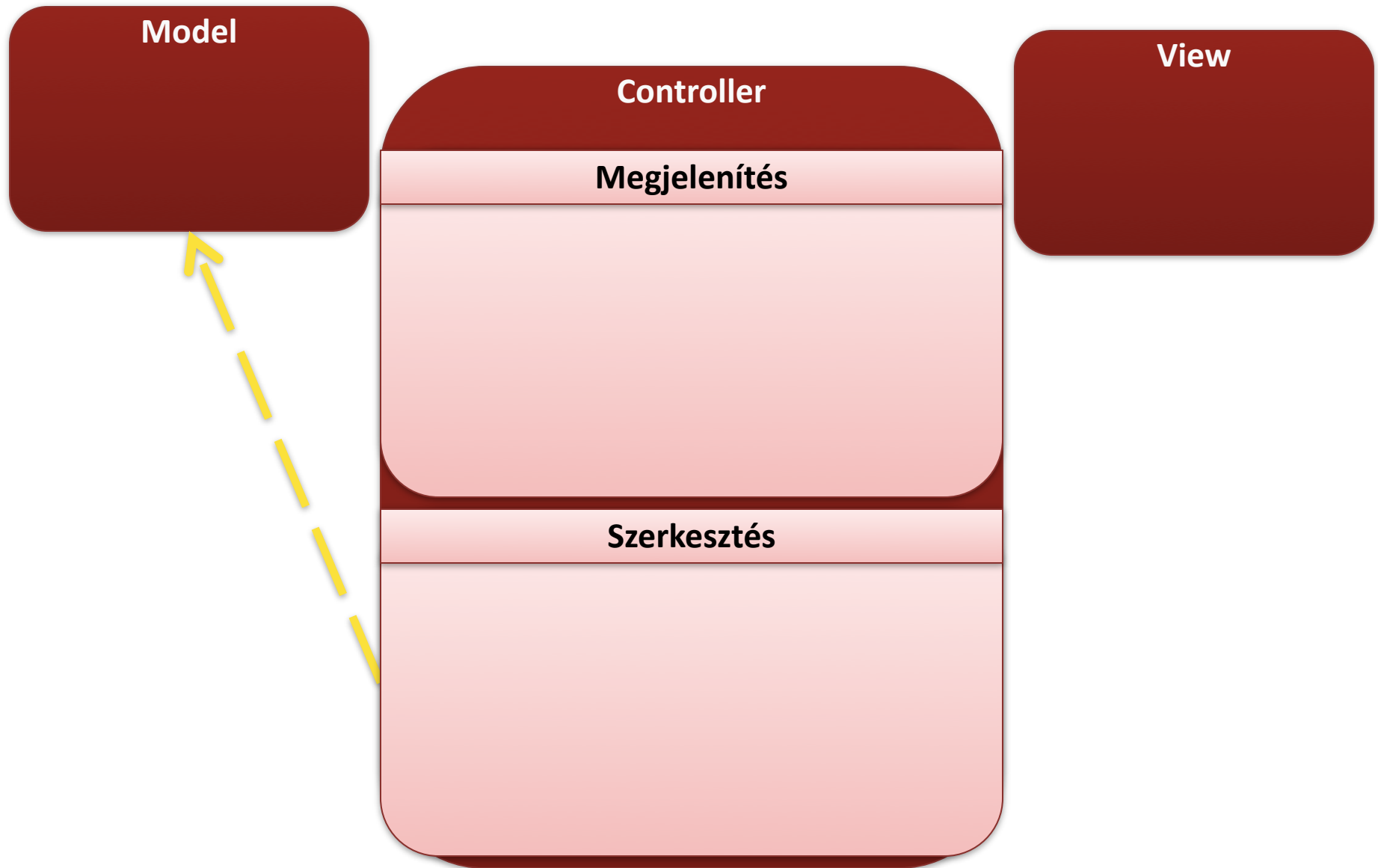
Modell ezt hívja

EditPart példa

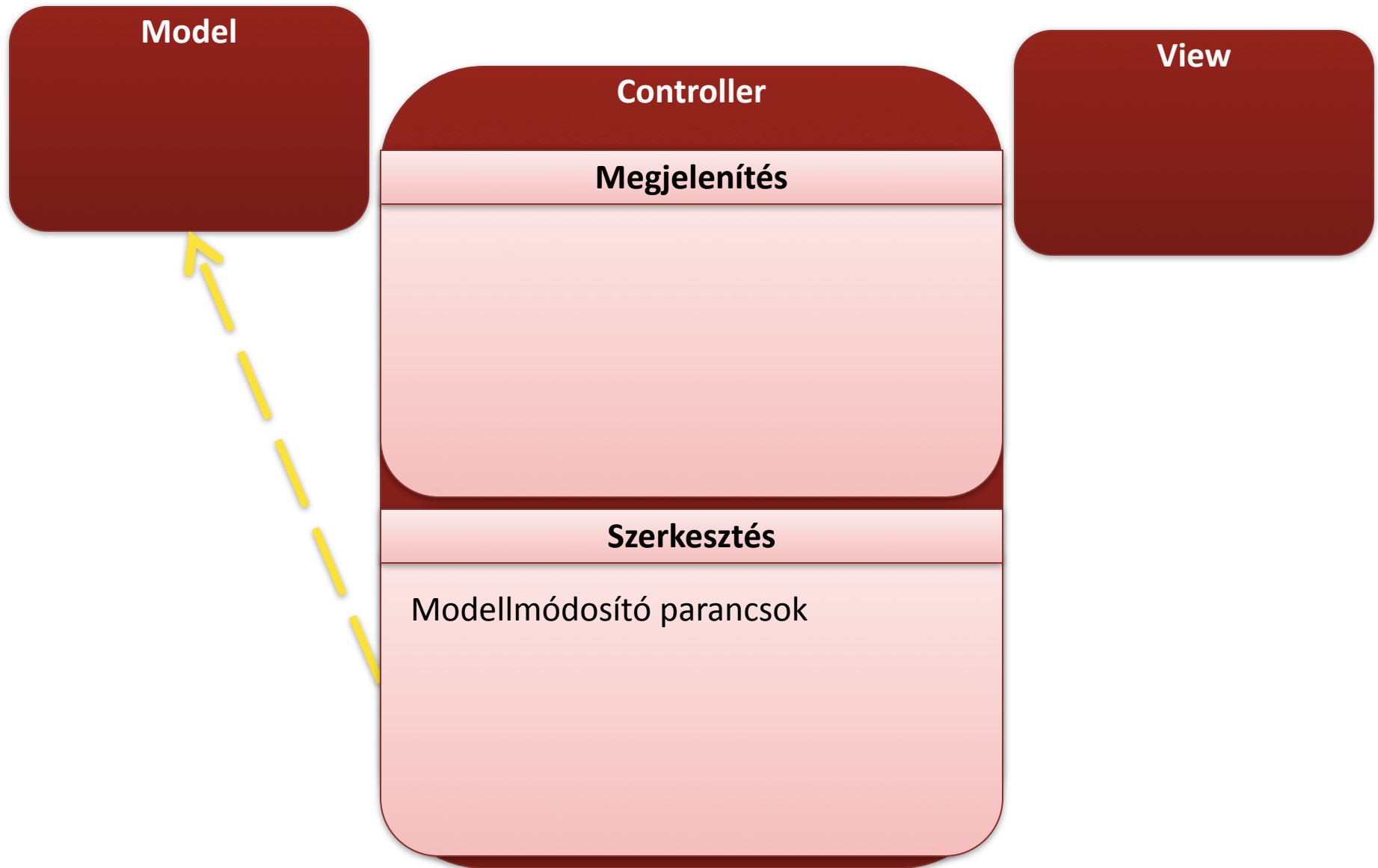
```
public class ParentEditPart extends AbstractGraphicalEditPart
    implements MyModelListener {
    protected void refreshVisuals() {
        ((ParentView) getFigure()).setLabel(
            ((ParentModel) getModel()).getName());
    }
    public void activate() {
        super.activate();
        ((ParentModel) getModel()).addListener(this);
    }
    public void deactivate() {
        ((ParentModel) getModel()).removeListener(this);
        super.deactivate();
    }
    public void modelChanged() {
        refreshVisuals();
        refreshChildren();
    }
}
```

Gyerekek
frissítése

GEF workflow



GEF workflow



GEF workflow

Model

Controller

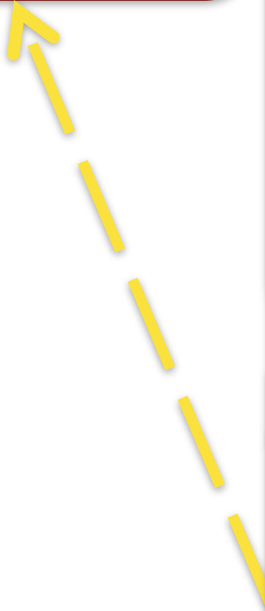
View

Megjelenítés

Szerkesztés

Modellmódosító parancsok

Szerkesztési kérések -> parancsok



GEF workflow

Model

Controller

View

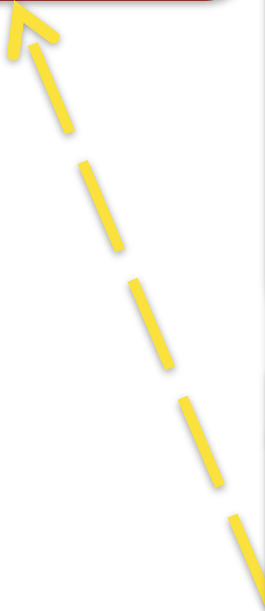
Megjelenítés

Szerkesztés

Modellmódosító parancsok

Szerkesztési kérések -> parancsok

Szerkesztőeszközök a felhasználói felületen



Szerkesztés szereplői I.

- EditDomain: fogadja az eseményeket az SWT-től, és továbbítja az aktív Toolnak
 - Nem végez feldolgozást, csak összefogja egy modell összes nézetét
- Tool: egy szerkesztési funkciót jelképez
 - Feldolgozza az SWT üzeneteket
 - Létrehoz egy (vagy több) Request-et

Szerkesztés szereplői II.

- Request
 - GEF-szintű esemény
 - Pl. CreateRequest, DeleteRequest
 - Továbbítódik a cél EditParthoz
- EditPolicy
 - EditParthoz tartozó „szerkesztési szabály”
 - Request -> Command leképezés
 - 1 EditPart -> több EditPolicy lehet

Szerkesztés szereplői III.

■ EditPart

- A saját EditPolicyjai segítségével átalakítja a bejövő Requestet egy Commandá
- Észleli a modell változását az értesítési mechanizmuson keresztül
- Modellváltozás esetén frissíti a nézetet, illetve a struktúrát

Szerkesztés szereplői IV.

- Command
 - A modell módosítását végzi
 - Visszavonható (ha megírjuk 😊)
- CommandStack
 - Végrehajtott Commandok verme
 - Ez biztosítja az undo/redo lehetőségét
 - EditDomainenként pontosan egy darab
 - Mindig ezen keresztül módosítsunk!

Szerkesztés szereplői V.

- Action
 - Nem GEF-specifikus (JFace)
 - Nem „grafikus” felhasználói akció
 - Menüelemek, billentyűlenyomások, toolbar elemek
 - GEF biztosít néhány wrappert, amik lehetővé teszik a CommandStack egyszerű elérését
 - ActionRegistry: actionök listája
 - Több helyen szereplő azonos actionökhöz
- Nincs több (lényeges) szereplő

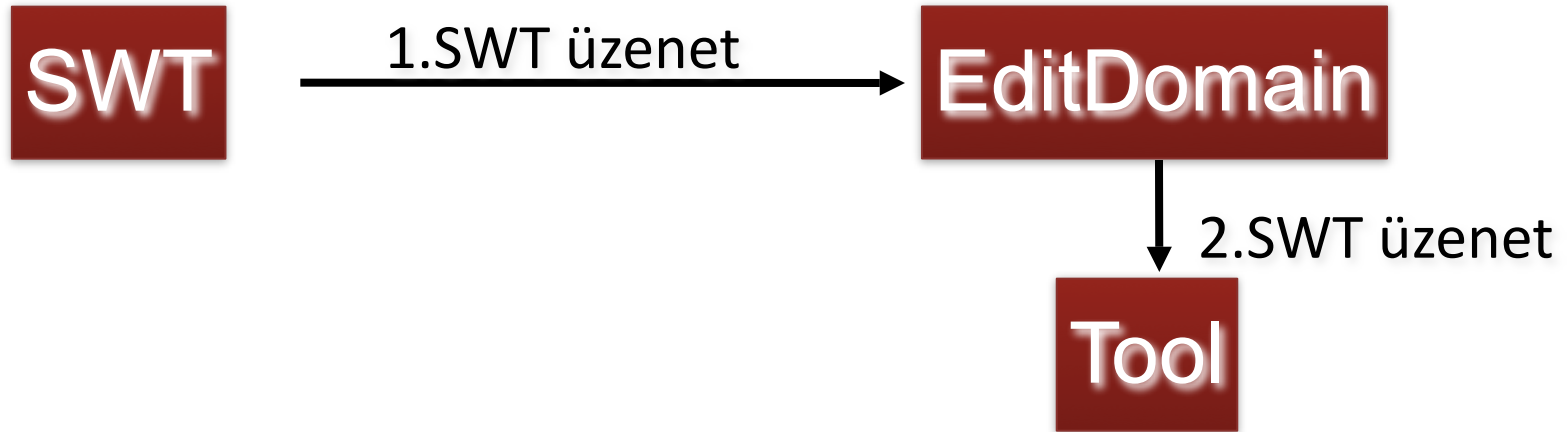
Szerkesztés folyamata

SWT

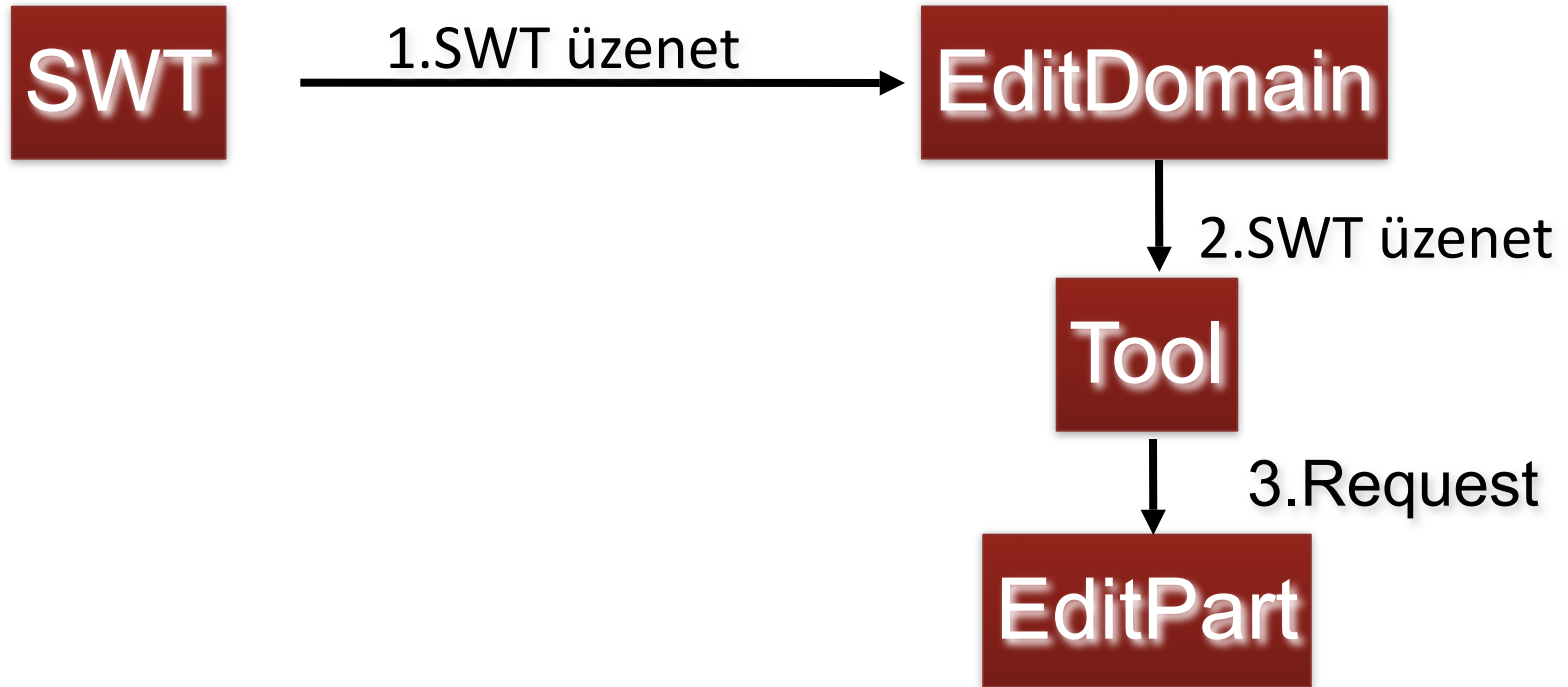
Szerkesztés folyamata



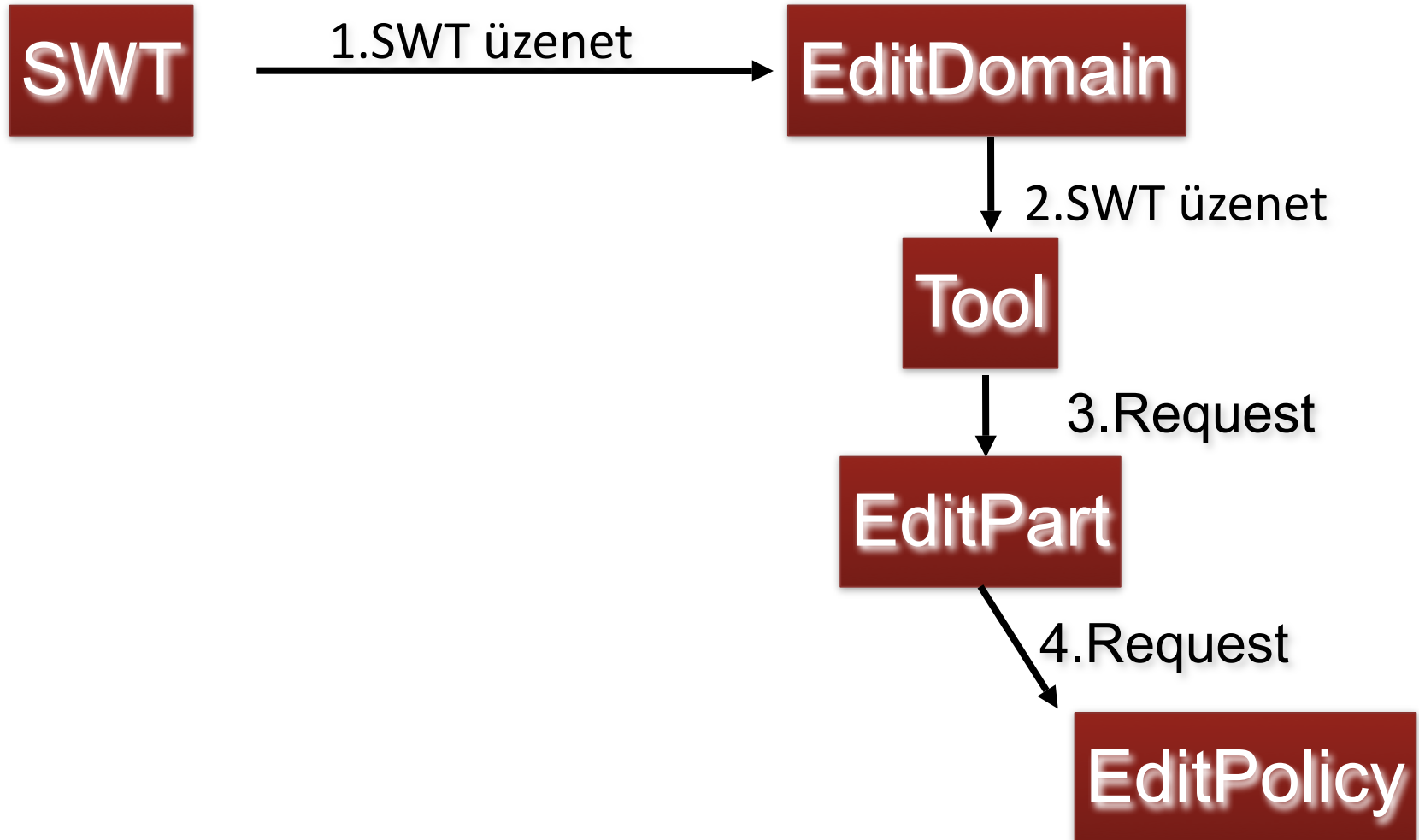
Szerkesztés folyamata



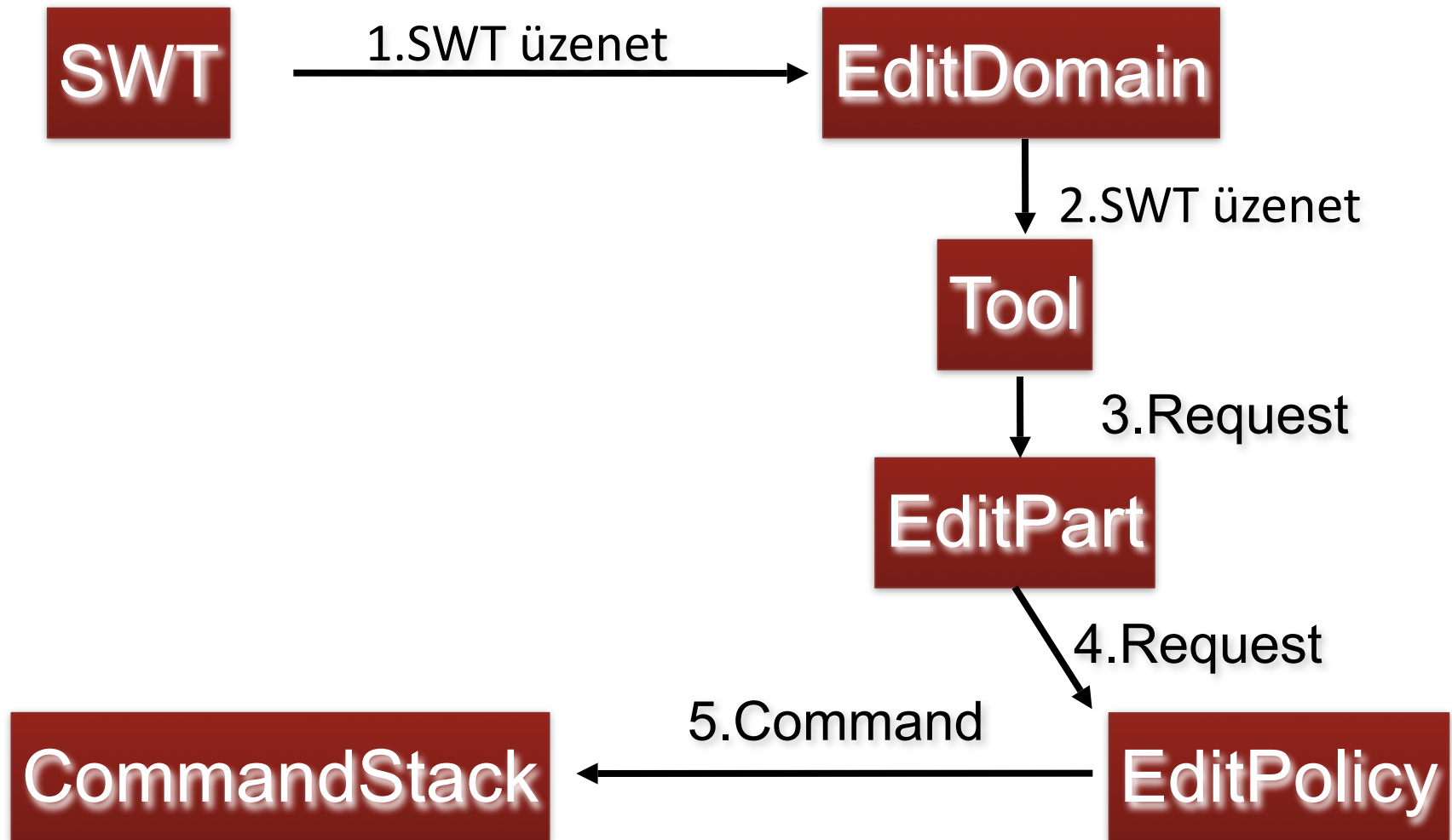
Szerkesztés folyamata



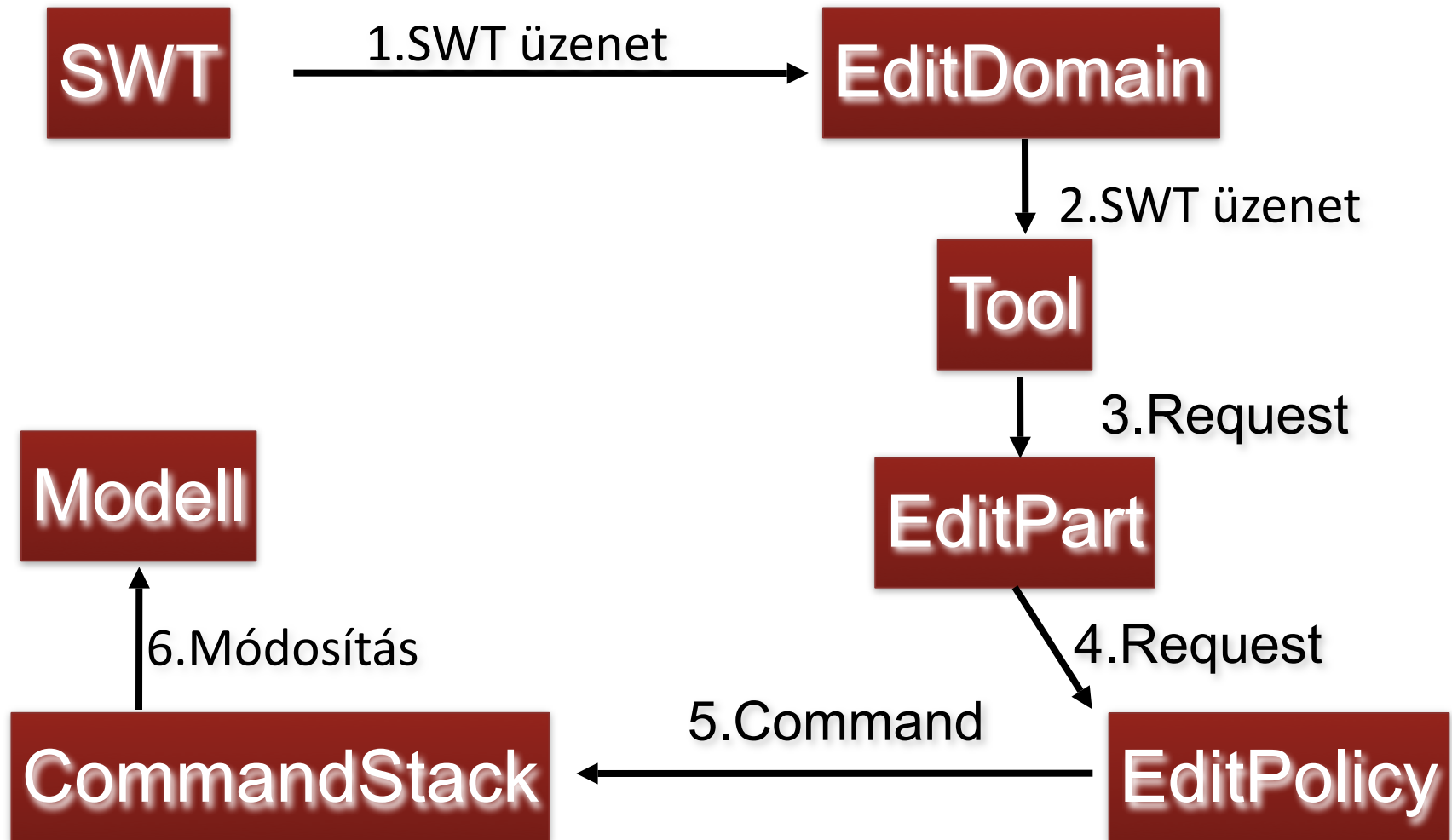
Szerkesztés folyamata



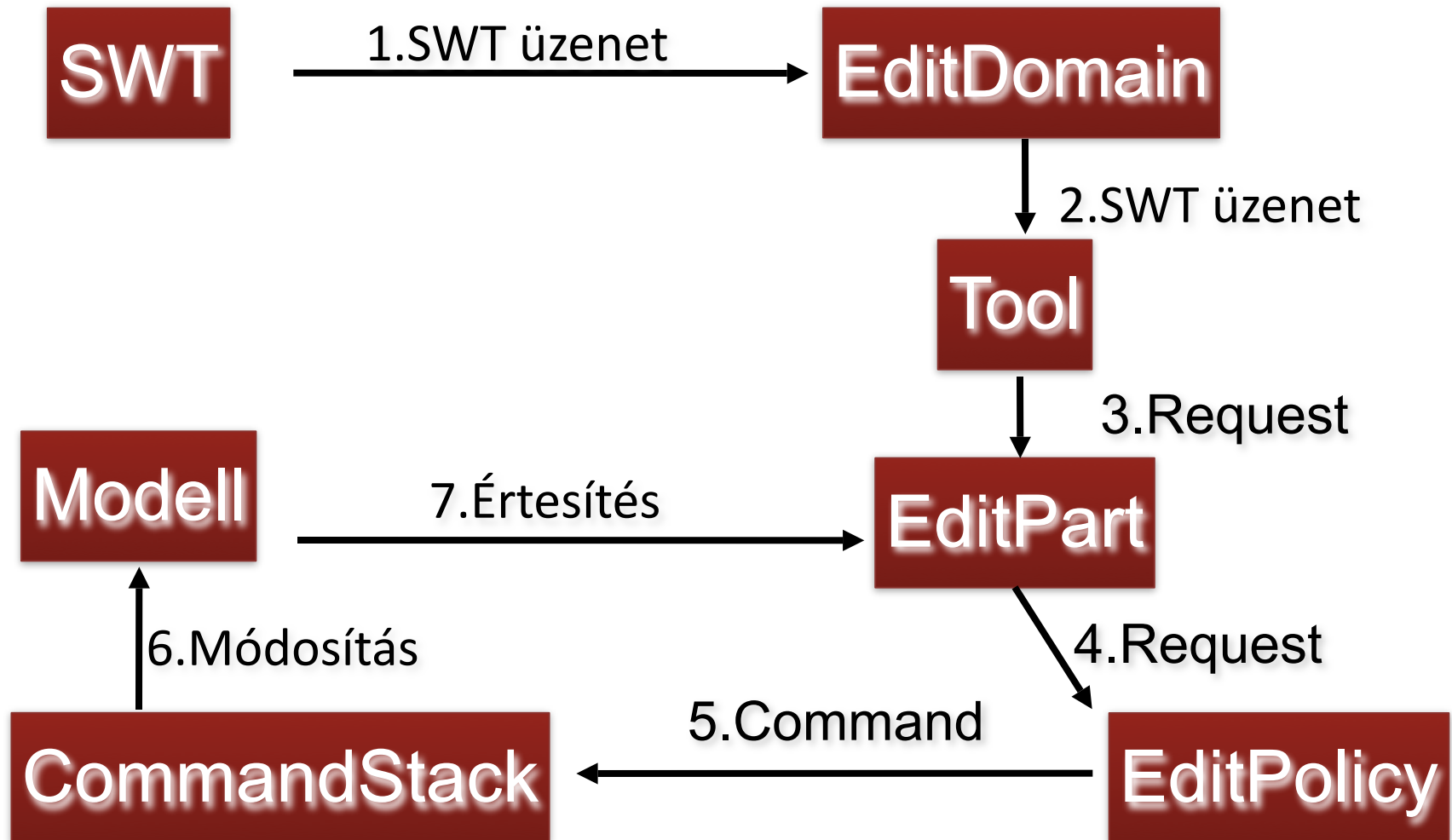
Szerkesztés folyamata



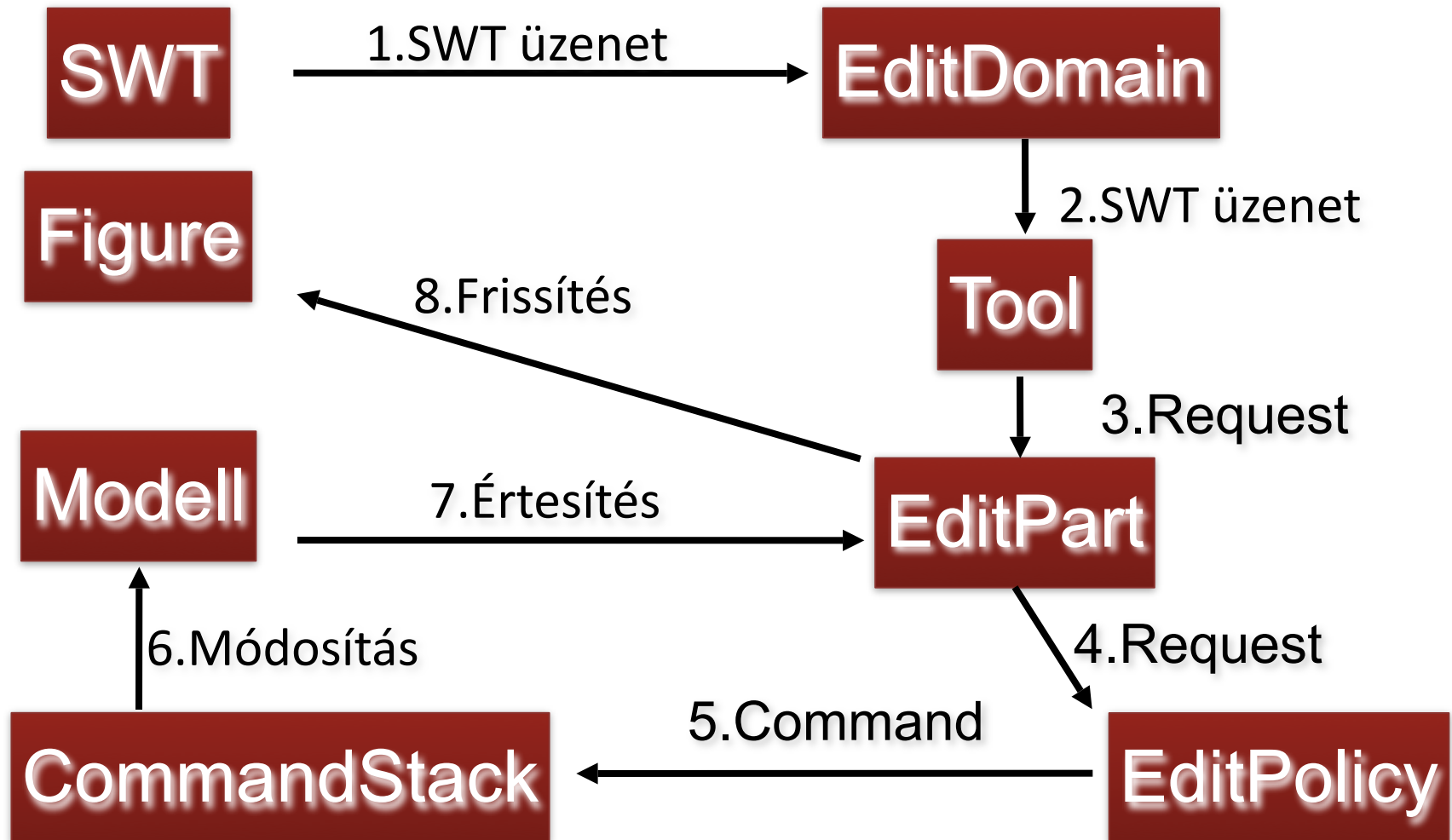
Szerkesztés folyamata



Szerkesztés folyamata



Szerkesztés folyamata



EditDomain

- GEF állapot
- Aktív eszköz (active tool)
 - Éppen használt szerkesztő funkció
 - Pl. kijelölés, új elem, törlés
- CommandStack
 - Elvégzett módosítások listája
 - Undo / redo támogatáshoz
- Használjuk mindig a DefaultEditDomain-t

Tool

- Beépített Toolok
 - SelectionTool, CreationTool, MarqueeTool
- Saját Tool is készíthető
 - TargetingTool: Ha van egy cél EditPart
 - AbstractTool: teljesen általános
- Aktív tool módosítása
 - EditDomain.setActiveTool()
 - Eszköztár (Palette): lásd később

Request

- ChangeBoundsRequest: átméretezés
- CreationRequest: elem létrehozása
- Minden Requesthez tartozik egy típus azonosító
 - RequestConstants osztályban
 - EditPolicy-k ez alapján azonosítják
 - Példák:
 - REQ_CREATE – létrehozás
 - REQ_DELETE – törlés
 - REQ_MOVE – mozgatás
 - REQ_RESIZE – átméretezés

EditPolicy

- Pontosan egy EditParthoz tartozik
 - getHost()-al lekérdezhető
 - 1 EditPart -> több EditPolicy lehet
- Szerep azonosítás: string kulcsokkal
 - EditPolicy osztályban konstansok, pl.:
 - COMPONENT_ROLE: alapvető műveletek (pl. törlés)
 - SELECTION_FEEDBACK_ROLE: visszacsatolás kijelölésnél
- Feladatai
 - Request -> Command leképezés
 - Command getCommand(Request)
 - Grafikus visszajelzés a felhasználónak
 - show(Source/Target)Feedback()

EditPolicy II.

- Beépített absztrakt őssosztályok
 - Némi előfeldolgozást végeznek a Requesten
 - ComponentEditPolicy: törlés
 - ContainerEditPolicy: létrehozás
 - LayoutEditPolicy: átméretezés
 - XYLayoutEditPolicy: átméretezés, ha az EditPart nézete XYLayoutot használ
 - Biztosítja a grafikus visszajelzést

EditPolicy példa

```
public class MyLayoutEditPolicy extends XYLayoutEditPolicy {
    protected Command createAddCommand(EditPart child,
        Object constraint) {
        return null;
    }

    protected Command createChangeConstraintCommand(
        EditPart child, Object constraint) {
        if (child.getModel() instanceof ElementModel &&
            constraint instanceof Rectangle)
            return new MyResizeCommand(((ElementModel)
                child.getModel()), ((Rectangle)
                    constraint));
        return null;
    }

    protected Command getCreateCommand(CreateRequest request) {
        return null;
    }

    protected Command getDeleteDependantCommand(Request req) {
        return null;
    }
}
```

EditPolicy példa

```
public class MyLayoutEditPolicy extends XYLayoutEditPolicy {
    protected Command createAddCommand(EditPart child,
        Object constraint) {
        return null;
    }
    protected Command createChangeConstraintCommand(
        EditPart child, Object constraint) {
        if (child.getModel() instanceof ElementModel &&
            constraint instanceof Rectangle)
            return new MyResizeCommand(((ElementModel)
                child.getModel()), ((Rectangle)
                    constraint));
        return null;
    }
    protected Command getCreateCommand(CreateRequest request) {
        return null;
    }
    protected Command getDeleteDependantCommand(Request req) {
        return null;
    }
}
```

XYLayout szülő

EditPolicy példa

```
public class MyLayoutEditPolicy extends XYLayoutEditPolicy {
    protected Command createAddCommand(EditPart child,
        Object constraint) {
        return null;
    }

    protected Command createChangeConstraintCommand(
        EditPart child, Object constraint) {
        if (child.getModel() instanceof ElementModel &&
            constraint instanceof Rectangle)
            return new MyResizeCommand(((ElementModel)
                child.getModel()), ((Rectangle)
                    constraint));
        return null;
    }

    protected Command getCreateCommand(CreateRequest request) {
        return null;
    }

    protected Command getDeleteDependantCommand(Request req) {
        return null;
    }
}
```

Átméretezés

EditPolicy példa

```
public class MyLayoutEditPolicy extends XYLayoutEditPolicy {
    protected Command createAddCommand(EditPart child,
        Object constraint) {
        return null;
    }

    protected Command createChangeConstraintCommand(
        EditPart child, Object constraint) {
        if (child.getModel() instanceof ElementModel &&
            constraint instanceof Rectangle)
            return new MyResizeCommand(((ElementModel)
                child.getModel()), ((Rectangle)
                    constraint));
        return null;
    }

    protected Command getCreateCommand(CreateRequest request) {
        return null;
    }

    protected Command getDeleteDependantCommand(EditPart child,
        Object constraint) {
        return null;
    }
}
```

Saját Command

Command példa

```
public class MyResizeCommand extends Command {
    ElementModel model;
    Rectangle newsize, oldsize;
    public MyResizeCommand(ElementModel m, Rectangle r) {
        model = m; newsize = r;
    }
    public boolean canExecute() {
        return (r.width >= 40 && r.height >= 40);
    }
    public void execute() {
        oldsize = model.getBounds();
        model.setBounds(newsize);
    }
    public boolean canUndo() {
        return true;
    }
    public void undo() {
        model.setBounds(oldsize);
    }
}
```


Command példa

```
public class MyResizeCommand extends Command {
    ElementModel model;
    Rectangle newsize, oldsize;
    public MyResizeCommand(ElementModel
        model = m; newsize = r;
    }
    public boolean canExecute() {
        return (r.width >= 40 && r.height >= 40);
    }
    public void execute() {
        oldsize = model.getBounds();
        model.setBounds(newsize);
    }
    public boolean canUndo() {
        return true;
    }
    public void undo() {
        model.setBounds(oldsize);
    }
}
```

Végrehajthatóság
feltétele

Command példa

```
public class MyResizeCommand extends Command {
    ElementModel model;
    Rectangle newsize, oldsize;
    public MyResizeCommand(ElementModel m, Rectangle r) {
        model = m; newsize = r;
    }
    public boolean canExecute() {
        return (r.width >= 40 && r.height >= 40);
    }
    public void execute() {
        oldsize = model.getBounds();
        model.setBounds(newsize);
    }
    public boolean canUndo() {
        return true;
    }
    public void undo() {
        model.setBounds(oldsize);
    }
}
```

Végrehajtás

Command példa

```
public class MyResizeCommand extends Command {
    ElementModel model;
    Rectangle newsize, oldsize;
    public MyResizeCommand(ElementModel m, Rectangle r) {
        model = m; newsize = r;
    }
    public boolean canExecute() {
        return (r.width >= 40 && r.height >= 40);
    }
    public void execute() {
        oldsize = model.getBounds();
        model.setBounds(newsize);
    }
    public boolean canUndo() {
        return true;
    }
    public void undo() {
        model.setBounds(oldsize);
    }
}
```

Visszavonhatóság
feltétele

Command példa

```
public class MyResizeCommand extends Command {
    ElementModel model;
    Rectangle newsize, oldsize;
    public MyResizeCommand(ElementModel m, Rectangle r) {
        model = m; newsize = r;
    }
    public boolean canExecute() {
        return (r.width >= 40 && r.height >= 40);
    }
    public void execute() {
        oldsize = model.getBounds();
        model.setBounds(newsize);
    }
    public boolean canUndo() {
        return true;
    }
    public void undo() {
        model.setBounds(oldsize);
    }
}
```

Visszavonás

EditPart és EditPolicy-k

■ EditPolicy-k telepítése

- `AbstractEditPart#createEditPolicies()` metódusban
- `EditPart#installEditPolicy(Object role, EditPolicy editPolicy)` metódus segítségével

```
public class ParentEditPart extends
    AbstractGraphicalEditPart {
    ...
    protected void createEditPolicies() {
        installEditPolicy(EditPolicy.LAYOUT_ROLE,
            new MyLayoutEditPolicy());
    }
    ...
}
```



EditPart és EditPolicy-k

■ EditPolicy-k telepítése

- `AbstractEditPart#createEditPolicies()` metódusban
- `EditPart#installEditPolicy(Object role, EditPolicy editPolicy)` metódus segítségével

```
public class ParentEditPart extends  
    AbstractGraphicalEditPart {  
    ...  
    protected void createEditPolicies() {  
        installEditPolicy(EditPolicy.LAYOUT_ROLE,  
new MyLayoutEditPolicy());  
    }  
    ...  
}
```

Policy telepítése



EditPart és EditPolicy-k

■ EditPolicy-k telepítése

- `AbstractEditPart#createEditPolicies()` metódusban
- `EditPart#installEditPolicy(Object role, EditPolicy editPolicy)` metódus segítségével

```
public class ParentEditPart extends
    AbstractGraphicalEditPart {
    ...
    protected void createEditPolicies() {
        installEditPolicy(EditPolicy.LAYOUT_ROLE,
            new MyLayoutEditPolicy());
    }
    ...
}
```



EditPart és EditPolicy-k

■ EditPolicy-k telepítése

- `AbstractEditPart#createEditPolicies()` metódusban
- `EditPart#installEditPolicy(Object role, EditPolicy editPolicy)` metódus segítségével

```
public class ParentEditPart extends
    AbstractGraphicalEditPart {
    ...
    protected void createEditPolicies() {
        installEditPolicy(EditPolicy.LAYOUT_ROLE,
            new MyLayoutEditPolicy());
    }
    ...
}
```

Szerep
azonosító



Mit kell nekünk megírni?

- Modell kód, értesítéssel
 - Generáltatható EMF segítségével
- Nézet osztályok
- EditPart osztályok 1.
 - Modell megjelenítés
 - createFigure(), refreshVisuals()
 - Modell változás figyelés
 - activate(), deactivate()
- EditPartFactory (modell -> EditPart)

Mit kell nekünk megírni?

- Modell módosító Commandok
- Saját EditPolicy-k, amik a Commandokat használják
 - Milyen műveleteket engedünk meg
- EditPart osztályok 2.
 - EditPolicy-k hozzárendelése
- Editor és tartozékai

Editor készítése

■ Feladatai

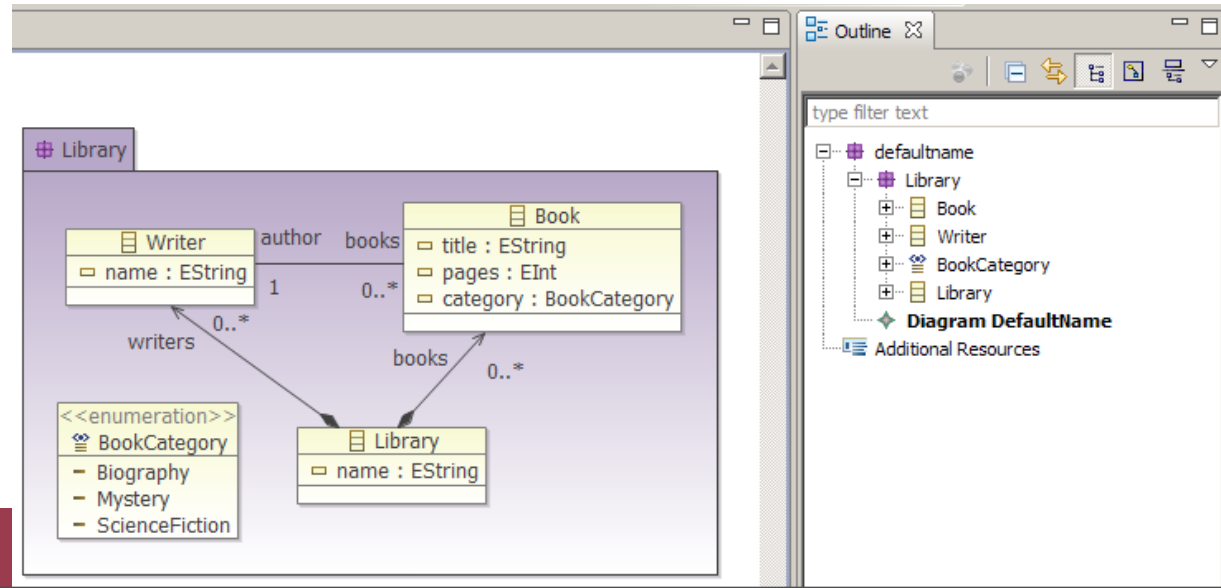
- Létrehoz egy EditPartViewert
- Kezeli a nem grafikus műveleteket
 - Actionök (undo/redo is ezek közé tartozik)
- Létrehozza a menü és toolbar bejegyzéseket
 - ActionBarContributor (lásd labor)

■ Megoldás

- Saját EditorPart, ezeket mi írjuk meg
- GraphicalEditor használata
 - Egyszerű, prototípushoz jó

EditPartViewer

- Egy EditPart hierarchia megjelenítéséért felelős
- Elvben hasonló, mint a JFace viewerek
- Fa- vagy grafikus nézet
 - TreeViewer: tipikusan Outline nézethez
 - GraphicalViewer: grafikus nézet
 - ScrollingGraphicalViewer: javasolt megvalósítás



EditPartViewer II.

- Három szükséges alkotóelem
 - EditDomain: GEF alkalmazás „állapota”
 - EditPartFactory: modell -> EditPart leképzés
 - Gyökér modellelem

```
public class TestGEFEditor extends EditorPart {
    ...
    public void createPartControl(Composite parent) {
        ScrollingGraphicalViewer viewer = new
        ScrollingGraphicalViewer();
        viewer.setEditDomain(new DefaultEditDomain
        (this));
        viewer.setEditPartFactory(new MyEditPartFactory
        ());
        viewer.setContents(modelRootElement);
        viewer.createControl(parent);
    }
    ...
}
```

GraphicalEditor

- Őosztály GEF-es Eclipse editorokhoz
 - Létrehoz egy ScrollingGraphicalViewer-t
 - Létrehoz egy pár általános Actiont
 - Undo, redo, törlés, nyomtatás, mentés
 - Nem jeleníti meg őket sehol
 - Használjuk az editor készítése közben teszteléshez, kísérletezéshez, de a végső alkalmazásba inkább ne kerüljön

GraphicalEditor használata

```
public class TestGEFEditor extends GraphicalEditor {
    public TestGEFEditor() {
        setEditDomain(new DefaultEditDomain(this));
    }

    protected void configureGraphicalViewer() {
        getGraphicalViewer().setEditPartFactory(
            new TestGEFEditPartFactory());
    }

    public void init(IEditorSite site, IEditorInput input)
        throws PartInitException {
        super.init(site, input);
        // Modell felépítése az input alapján
    }

    protected void initializeGraphicalViewer() {
        getGraphicalViewer().setContents(modelRoot);
    }

    ...
}
```

GraphicalEditor használata

EditDomain a konstruktorban

```
public class TestGEFEditor extends GraphicalEditor {
    public TestGEFEditor() {
        setEditDomain(new DefaultEditDomain(this));
    }

    protected void configureGraphicalViewer() {
        getGraphicalViewer().setEditPartFactory(
            new TestGEFEditPartFactory());
    }

    public void init(IEditorSite site, IEditorInput input)
        throws PartInitException {
        super.init(site, input);
        // Modell felépítése az input alapján
    }

    protected void initializeGraphicalViewer() {
        getGraphicalViewer().setContents(modelRoot);
    }

    ...
}
```


GraphicalEditor használata

```
public class TestGEFEditor extends GraphicalEditor {
    public TestGEFEditor() {
        setEditDomain(new DefaultEditDomain(this));
    }
    protected void configureGraphicalViewer() {
        getGraphicalViewer().setEditPartFactory(
            new TestGEFEditPartFactory());
    }
    public void init(IEditorSite site, IEditorInput input)
        throws PartInitException {
        super.init(site, input);
        // Modell felépítése az input alapján
    }
    protected void initializeGraphicalViewer() {
        getGraphicalViewer().setContents(modelRoot);
    }
    ...
}
```

EditPartFactory
megadása

GraphicalEditor használata

```
public class TestGEFEditor extends GraphicalEditor {
    public TestGEFEditor() {
        setEditDomain(new DefaultEditDomain(this));
    }
    protected void configureGraphicalViewer() {
        getGraphicalViewer().setEditPartFactory(
            new TestGEFEditPartFactory());
    }
    public void init(IEditorSite site, IEditorInput input)
        throws PartInitException {
        super.init(site, input);
        // Modell felépítése az input alapján
    }
    protected void initializeGraphicalViewer() {
        getGraphicalViewer().setContents(modelRoot);
    }
    ...
}
```

Megnyitott fájl
feldolgozása
(Eclipse editor)

GraphicalEditor használata

```
public class TestGEFEditor extends GraphicalEditor {
    public TestGEFEditor() {
        setEditDomain(new DefaultEditDomain(this));
    }
    protected void configureGraphicalViewer() {
        getGraphicalViewer().setEditPartFactory(
            new TestGEFEditPartFactory());
    }
    public void init(IEditorSite site, IEditorInput input)
        throws PartInitException {
        super.init(site, input);
        // Modell felépítése az input alapján
    }
    protected void initializeGraphicalViewer() {
        getGraphicalViewer().setContents(modelRoot);
    }
    ...
}
```

Modell
gyökérelem
megadása

Eszköztár (Palette)

- Aktív eszköz váltása
- Eszközök grafikus megjelenítése
- Belül ez is egy külön GEF GraphicalViewer
- PaletteRoot: eszköztár gyökere
- PaletteEntry: eszköztár bejegyzés
 - PaletteContainer: eszközök csoportja
 - ToolEntry: egy konkrét eszköz

Gyakori ToolEntry-k

- SelectionToolEntry: kijelölés eszköz
- MarqueeToolEntry: csoportos kijelölés
- CreationToolEntry: elem létrehozása
 - Nehézkes használni
 - Factory osztály, ez lekérdezhető a Requesten keresztül az EditPolicyban -> azonosítás
- Minden ToolEntry-hoz tartozik
 - Név, rövid leírás, kis/nagy ikon
 - Tool osztály, amit példányosít

GraphicalEditorWithPalette

- Olyan GraphicalEditor, ami létrehozza saját magának az eszköztárat
- Szintén csak prototípushoz javasolt használni
- Eszköztárhoz csak egy függvényt kell megírunk
 - `getPaletteRoot()`

GraphicalEditorWithPalette példa

```
public class TestGEFEditor extends GraphicalEditorWithPalette {
    protected PaletteRoot getPaletteRoot() {
        PaletteRoot root = new PaletteRoot();
        PaletteGroup selectionToolGroup = new PaletteGroup
("Selection");
        ToolEntry tool = new SelectionToolEntry();
        selectionToolGroup.add(tool);
        root.setDefaultEntry(tool);
        tool = new MarqueeToolEntry();
        selectionToolGroup.add(tool);
        root.add(selectionToolGroup);
        root.add(new PaletteSeparator());
        root.add(new CreationToolEntry("New Place",
            "Creates a new Petri net place",
            new SimpleFactory(PetriPlace.class),
            MyPlugin.getImageDescriptor("place.png"),
            MyPlugin.getImageDescriptor("place.png")));
        ...
        return root;
    }
    ...
}
```

GraphicalEditorWithPalette

Új eszköztár

```
public class TestGEFEditor extends GraphicalEditor {
    protected PaletteRoot getPaletteRoot() {
        PaletteRoot root = new PaletteRoot();
        PaletteGroup selectionToolGroup = new PaletteGroup
("Selection");
        ToolEntry tool = new SelectionToolEntry();
        selectionToolGroup.add(tool);
        root.setDefaultEntry(tool);
        tool = new MarqueeToolEntry();
        selectionToolGroup.add(tool);
        root.add(selectionToolGroup);
        root.add(new PaletteSeparator());
        root.add(new CreationToolEntry("New Place",
            "Creates a new Petri net place",
            new SimpleFactory(PetriPlace.class),
            MyPlugin.getImageDescriptor("place.png"),
            MyPlugin.getImageDescriptor("place.png")));
        ...
        return root;
    }
    ...
}
```


GraphicalEditorWithPalette példa

```
public class TestGEFEditor extends GraphicalEditorWithPalette {  
    protected PaletteRoot getPaletteRoot() {  
        PaletteRoot root = new PaletteRoot();  
        PaletteGroup selectionToolGroup = new PaletteGroup  
("Selection");  
        ToolEntry tool = new SelectionToolEntry();  
        selectionToolGroup.add(tool);  
        root.setDefaultEntry(tool);  
        tool = new MarqueeToolEntry();  
        selectionToolGroup.add(tool);  
        root.add(selectionToolGroup);  
        root.add(new PaletteSeparator());  
        root.add(new CreationToolEntry("New Place",  
            "Creates a new Petri net place",  
            new SimpleFactory(PetriPlace.class),  
            MyPlugin.getImageDescriptor("place.png"),  
            MyPlugin.getImageDescriptor("place.png")));  
        ...  
        return root;  
    }  
    ...  
}
```

Új csoport

GraphicalEditorWithPalette példa

```
public class TestGEFEditor extends GraphicalEditorWithPalette {
    protected PaletteRoot getPaletteRoot() {
        PaletteRoot root = new PaletteRoot();
        PaletteGroup selectionToolGroup = new PaletteGroup
("Selection");
        ToolEntry tool = new SelectionToolEntry();
        selectionToolGroup.add(tool);
        root.setDefaultEntry(tool);
        tool = new MarqueeToolEntry();
        selectionToolGroup.add(tool);
        root.add(selectionToolGroup);
        root.add(new PaletteSeparator());
        root.add(new CreationToolEntry("New Place",
            "Creates a new Petri net place",
            new SimpleFactory(PetriPlace.class),
            MyPlugin.getImageDescriptor("place.png"),
            MyPlugin.getImageDescriptor("place.png")));
        ...
        return root;
    }
    ...
}
```

GraphicalEditorWithPalette példa

```
public class TestGEFEditor extends GraphicalEditorWithPalette {
    protected PaletteRoot getPaletteRoot() {
        PaletteRoot root = new PaletteRoot();
        PaletteGroup selectionToolGroup = new PaletteGroup
("Selection");
        ToolEntry tool = new SelectionToolEntry();
        selectionToolGroup.add(tool);
        root.setDefaultEntry(tool);
        tool = new MarqueeToolEntry();
        selectionToolGroup.add(tool);
        root.add(selectionToolGroup);
        root.add(new PaletteSeparator());
        root.add(new CreationToolEntry("New Place",
            "Creates a new Petri net place",
            new SimpleFactory(PetriPlace.class),
            MyPlugin.getImageDescriptor("place.png"),
            MyPlugin.getImageDescriptor("place.png"));
        ...
        return root;
    }
    ...
}
```

Elválasztó vonal

GraphicalEditorWithPalette példa

```
public class TestGEFEditor extends GraphicalEditorWithPalette {
    protected PaletteRoot getPaletteRoot() {
        PaletteRoot root = new PaletteRoot();
        PaletteGroup selectionToolGroup = new PaletteGroup
("Selection");
        ToolEntry tool = new SelectionToolEntry();
        selectionToolGroup.add(tool);
        root.setDefaultEntry(tool);
        tool = new MarqueeToolEntry();
        selectionToolGroup.add(tool);
        root.add(selectionToolGroup);
        root.add(new PaletteSeparator());
        root.add(new CreationToolEntry("New Place",
            "Creates a new Petri net place",
            new SimpleFactory(PetriPlace.class),
            MyPlugin.getImageDescriptor("place.png"),
            MyPlugin.getImageDescriptor("place.png")));
        ...
        return root;
    }
    ...
}
```

Factory a tool
létrehozáshoz

GraphicalEditorWithPalette példa

```
public class TestGEFEditor extends GraphicalEditorWithPalette {
    protected PaletteRoot getPaletteRoot() {
        PaletteRoot root = new PaletteRoot();
        PaletteGroup selectionToolGroup = new PaletteGroup
("Selection");
        ToolEntry tool = new SelectionToolEntry();
        selectionToolGroup.add(tool);
        root.setDefaultEntry(tool);
        tool = new MarqueeToolEntry();
        selectionToolGroup.add(tool);
        root.add(selectionToolGroup);
        root.add(new PaletteSeparator());
        root.add(new CreationToolEntry("New Place",
            "Creates a new Petri net place",
            new SimpleFactory(PetriPlace.class),
            MyPlugin.getImageDescriptor("place.png"),
            MyPlugin.getImageDescriptor("place.png")));
        ...
        return root;
    }
    ...
}
```

GEF workflow

Model

Értesítés

Controller

Megjelenítés

Kapcsolat a
modellel

Kapcsolat a
nézettel

Hierarchia megadása

Nézet frissítése a modell változásakor

Szerkesztés

Modellmódosító parancsok

Szerkesztési kérések -> parancsok

Szerkesztőeszközök a felhasználói
felületen

View

Kirajzolás

Elrendezés

GEF workflow

Model

Értesítés

EditPartFactory

Megjelenítés

Kapcsolat a
modellel

Kapcsolat a
nézettel

Hierarchia megadása

Nézet frissítése a modell változásakor

Szerkesztés

Modellmódosító parancsok

Szerkesztési kérések -> parancsok

Szerkesztőeszközök a felhasználói
felületen

View

Kirajzolás

Elrendezés

GEF workflow

Figure

Model

Értesítés

Controller

Megjelenítés

Kapcsolat a
modellel

Kapcsolat a
nézettel

Hierarchia megadása

Nézet frissítése a modell változásakor

Szerkesztés

Modellmódosító parancsok

Szerkesztési kérések -> parancsok

Szerkesztőeszközök a felhasználói
felületen

ew

Kirajzolás

Elrendezés

GEF workflow

Model

Értesítés

Controller

Megjelenítés

Kapcsolat a
modellel

Kapcsolat a
nézettel

Hierarchia megadása

Nézet frissítése a modell változásakor

Szerkesztés

Modellmódosító parancsok

Szerkesztési kérések -> parancsok

Szerkesztőeszközök a felhasználói
felületen

View

Kirajzolás

Elrendezés

LayoutManager

GEF workflow

Model

Értesítés

Controller

Megjelenítés

Kapcsolat a
modellel

Kapcsolat a
nézettel

Hierarchia megadása

Nézet frissítése a modell változásakor

Szerkesztés

Modellmódosító parancsok

Szerkesztési kérések -> parancsok

Szerkesztőeszközök a felhasználói
felületen

View

Kirajzolás

Elrendezés

EditPart

GEF workflow

Model

Értesítés

Controller

Megjelenítés

Kapcsolat a
modellel

Kapcsolat a
nézettel

Hierarchia megadása

Nézet frissítése a modell változásakor

Szerkesztés

Modellmódosító parancsok

Szerkesztési kérések -> parancsok

Szerkesztőeszközök a felhasználói
felületen

View

Kirajzolás

Elrendezés

Command

GEF workflow

Model

Értesítés

Controller

Megjelenítés

Kapcsolat a
modellel

Kapcsolat a
nézettel

Hierarchia megadása

Nézet frissítése a modell változásakor

Szerkesztés

Modellmódosító parancsok

Szerkesztési kérések -> parancsok

Szerkesztőeszközök a felhasználói
felületen

View

Kirajzolás

Elrendezés

EditPolicy

GEF workflow

Model

Értesítés

Controller

Megjelenítés

Kapcsolat a
modellel

Kapcsolat a
nézettel

Hierarchia megadása

Nézet frissítése a modell változásakor

Szerkesztés

Modellmódosító parancsok

Szerkesztési kérések -> parancsok

Szerkesztőeszközök a felhasználói
felületen

View

Kirajzolás

Elrendezés

Tool

GEF workflow

Model

Értesítés

Controller

Megjelenítés

Kapcsolat a
modellel

Kapcsolat a
nézettel

Hierarchia megadása

Nézet frissítése a modell változásakor

Szerkesztés

Modellmódosító parancsok

Szerkesztési kérések -> parancsok

Szerkesztőeszközök a felhasználói
felületen

View

Kirajzolás

Elrendezés

Összekötők használata a GEF-ben

Nyilak (összekötők)

- Hasonlóak a normál objektumokhoz
 - DE: fontos különbségek
- Megjelenítés külön (felsőbb) rétegben
- Van saját EditPart
 - AbstractConnectionEditPartból származik
 - Saját EditPolicy-k, Requestek, stb.
- Irányítottak (modell szinten)

GEF workflow

Model

Forrás, cél elérése

Kirajzolás

Elrendezés

Nyíl modell

- Szintén semmi megkötés
- Két lehetőség:
 - Osztály reprezentálja
 - Attribútum reprezentálja
- Tudnia kell a saját forrását és célját

GEF workflow

Model

View

Kirajzolás

Elrendezés

Nyíl nézet

- Draw2D PolylineConnection példány
- Figure leszármazott -> lehetnek gyerekei
- GEF-ben nincs nyíl hierarchia
 - Mindegyik a teljes szerkesztőt kitöltheti
- Speciális elemek
 - ConnectionAnchor: végpontok helye
 - ConnectionRouter: nyíl alakja
 - RotatableDecoration: végpontok „dísze”

ConnectionAnchor

- Összekötők végpontjai
- Két megvalósítás
 - ChopboxAnchor: téglalap Figure-höz
 - EllipseAnchor: ellipszis Figure-höz
- Egyéb esetben kell sajátot írni

ConnectionRouter

- A két Anchor közötti közbenső pontokat számolja ki
- \approx LayoutManager, itt is lehet Constraint
- Típusai
 - NullRouter: egyenes vonal
 - BendpointConnectionRouter: kézi töréspontok
 - ManhattanConnectionRouter: automatikus derékszögű töréspontok
 - ShortestPathConnectionRouter: összekötő kikerüli az akadályokat
- Görbe nyilak támogatása
 - GEF-ben nincs beépítve, saját osztállyal kell megoldani

Nyíl nézet példa

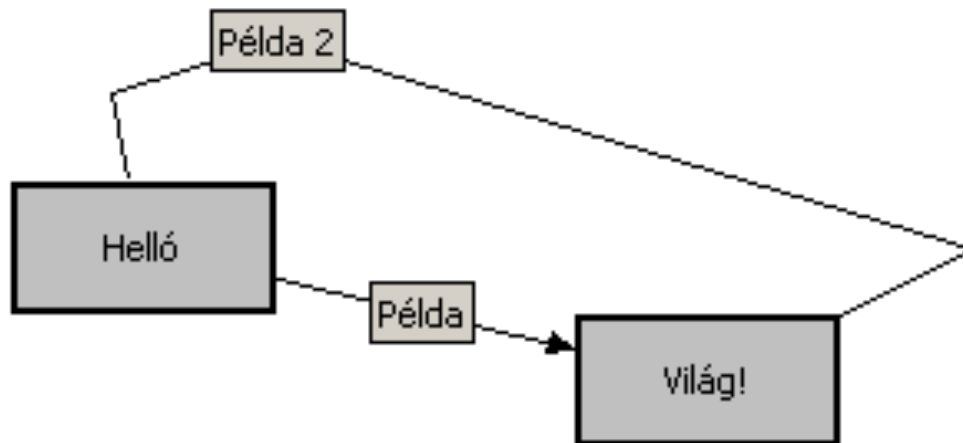
```
public class TestConnectionView extends PolylineConnection {
    private Label label;

    public TestConnectionView() {
        label = new Label();
        label.setOpaque(true);
        label.setBorder(new LineBorder());
        add(label, new ConnectionLocator(this,
            ConnectionLocator.MIDDLE));

        PolygonDecoration decoration = new PolygonDecoration();
        decoration.setTemplate(PolygonDecoration.TRIANGLE_TIP);
        setTargetDecoration(decoration);

        setConnectionRouter(new BendpointConnectionRouter());
    }

    public setBendpoints(List<Bendpoint> bendpoints) {
        setRoutingConstraint(bendpoints);
    }
}
```



GEF workflow

Model

Controller

View

Megjelenítés

Kapcsolódó modell Végpontok
összekötők

Forrás és cél megadása

Kapcsolat a modellel

- Forrás/cél objektumnak mindenképpen tudnia kell a nyilakról
- Navigálás a nyilakhoz a forrás/cél AbstractGraphicalEditPartban
- getModel(Source | Target)Connections()

...

```
protected List getModelSourceConnections() {  
    return ((ElementModel)getModel()).getSourceConnections();  
}
```

```
protected List getModelTargetConnections() {  
    return ((ElementModel)getModel()).getTargetConnections();  
}
```

...

Kapcsolat a nézettel - NodeEditPart

- ConnectionAnchorok visszaadása
- Forrás/cél EditPartok biztosítják
 - NodeEditPart interfészben deklarált metódusok
 - `get(Source|Target)ConnectionAnchor()`

Nyíl EditPart

- **Ősosztály: AbstractConnectionEditPart**
 - Mindent tud, amit a többi EditPart
- **Nyíl nézet létrehozása: createFigure()**
 - Mindenképpen egy PolylineConnection kell
- **Forrás és cél EditPartok lekérdezhetők**
 - getSource() és getTarget() függvények
 - Létrehozás közben nem feltétlenül elérhető!
- **Szerepelnie kell az EditPartFactoryban is**

GEF workflow

Model

Controller

View

Megjelenítés

Szerkesztés

Összeköttetés elkezdéséhez és
befejezéséhez tartozó parancsok

Egy műveletekhez tartozó parancsok

EditPolicy: Nyíl létrehozása

■ GraphicalNodeEditPolicy

- Nem a nyíl EditParthoz, hanem a forrás/cél elemek EditPartjához tartozik!
- Role: GRAPHICAL_NODE_ROLE

■ Két lépcsős létrehozás

- 1. getConnectionCreateCommand()
 - Request -> köztes Command (ez nem hajtódik végre)
- 2. getConnectionCompleteCommand()
 - Request + köztes Command -> végső Command
 - Csak ez hajtódik végre!

Nyíl EditPolicy-k

- Speciális nyíl EditPolicy-k
 - ConnectionEndpointEditPolicy: kijelölés
 - Role: CONNECTION_ENDPOINTS_ROLE
 - ConnectionEditPolicy: törlés
 - Role: CONNECTION_ROLE
 - BendpointEditPolicy: töréspontok módosítása
 - Role: CONNECTION_BENDPOINTS_ROLE
 - Csak akkor, ha BendpointConnectionRouter van

Mit kell nekünk megírni?

- Nyíl modell kód, értesítéssel
 - Nyíl források/célok modelljében el kell tudni érni az onnan induló/oda érkező nyilakat!
- Nyíl nézet osztályok
- Forrás/cél EditPart osztályok
 - Kapcsolódó kimenő/bejövő összekötők modellbeli reprezentációja
 - getSourceConnections(), getTargetConnections()
 - Végpontok
 - getSourceConnectionAnchor(), getTargetConnectionAnchor()
- Nyíl EditPart osztályok
 - Forrás/cél visszaadása
 - getSource(), getTarget()

Mit kell nekünk megírni?

- Nyíl szerkesztő Commandok
- Nyíl létrehozás EditPolicy
 - Forrás/cél EditParthoz tartozik
 - GraphicalNodeEditPolicy
 - Két lépcsős
- Nyíl saját EditPolicy-k
 - Törlés, töréspontok módosítása stb.

GEF workflow

Model

Forrás, cél elérése

Controller

Megjelenítés

Kapcsolódó modell Végpontok
összekötők

Forrás és cél megadása

Szerkesztés

Összeköttetés elkezdéséhez és
befejezéséhez tartozó parancsok

Egy műveletekhez tartozó parancsok

View

Kirajzolás

Elrendezés

GEF workflow

Model

Forrás, cél elérése

PolylineConnection
PolylineDecoration

View

Kirajzolás

Elrendezés

Megjelenítés

Kapcsolódó modell Végpontok
összekötők

Forrás és cél megadása

Szerkesztés

Összeköttetés elkezdéséhez és
befejezéséhez tartozó parancsok

Egy műveletekhez tartozó parancsok

GEF workflow

Model

Forrás, cél elérése

Controller

Megjelenítés

Kapcsolódó modell Végpontok
összekötők

Forrás és cél megadása

Szerkesztés

Összeköttetés elkezdéséhez és
befejezéséhez tartozó parancsok

Egy műveletekhez tartozó parancsok

View

Kirajzolás

Elrendezés

ConnectionAnchor
ConnectionRouter

GEF workflow

Model

Forrás, cél elérése

GraphicalEditPart

Controller

Megjelenítés

Kapcsolódó modell összekötők Végpontok

Forrás és cél megadása

Szerkesztés

Összeköttetés elkezdéséhez és befejezéséhez tartozó parancsok

Egy műveletekhez tartozó parancsok

View

Kirajzolás

Elrendezés

GEF workflow

Model

Forrás, cél elérése

Controller

Megjelenítés

Kapcsolódó modell összekötők Végpontok

Forrás és cél megadása

Szerkesztés

Összeköttetés elkezdéséhez és befejezéséhez tartozó parancsok

Egy műveletekhez tartozó parancsok

View

Kirajzolás

Elrendezés

NodeEditPart

GEF workflow

Model

Forrás, cél elérése

Controller

Megjelenítés

Kapcsolódó modell összekötők Végpontok

Forrás és cél megadása

Szerkesztés

Összeköttetés elkezdéséhez és befejezéséhez tartozó parancsok

Egy műveletekhez tartozó parancsok

View

Kirajzolás

Elrendezés

ConnectionEditPart

GEF workflow

Model

Forrás, cél elérése

Controller

Megjelenítés

Kapcsolódó modell Végpontok
összekötők

Forrás és cél megadása

Szerkesztés

Összeköttetés elkezdéséhez és
befejezéséhez tartozó parancsok

Egy műveletekhez tartozó parancsok

View

Kirajzolás

Elrendezés

GraphNodeE
ditPolicy

GEF workflow

Model

Forrás, cél elérése

Controller

Megjelenítés

Kapcsolódó modell Végpontok
összekötők

Forrás és cél megadása

Szerkesztés

Összeköttetés elkezdéséhez és
befejezéséhez tartozó parancsok

Egy műveletekhez tartozó parancsok

View

Kirajzolás

Elrendezés

Haladó GEF fejlesztés

További lehetőségek

- Modell tulajdonságok szerkesztése az Eclipse *Properties* nézetében
- Szövegek (címkék) szerkesztése közvetlenül a rajzon (direct editing)
- Nagyítási lehetőség
- Igazítás
- Különálló fa és áttekintő modellnézet

Modell tulajdonságok

- A beépített Properties nézetet használjuk
- Mindig az aktuálisan kijelölt EditParthoz tartozó tulajdonságok jelennek meg
 - Csoportokba szervezhetőek
- API: `org.eclipse.ui.views.properties`
- Szerkesztés beépített `PropertySheetPage` implementációval

Modell tulajdonságok

■ Tulajdonságok leírása

○ IPropertySource interfész

- getPropertyDescriptors()
 - IPropertyDescriptor tömb (!)
- getPropertyValue()
- setPropertyValue()
- isPropertySet()
 - Különbözik-e a jelenlegi érték a default-tól
- resetPropertyValue()
- getEditableValue()
 - A szerkesztőben más is megjeleníthető, pl. fully qualified name helyett local name

○ IPropertySource2 interfész

- isPropertyResettable(): visszaállítható-e egy értelmes

Modell tulajdonságok

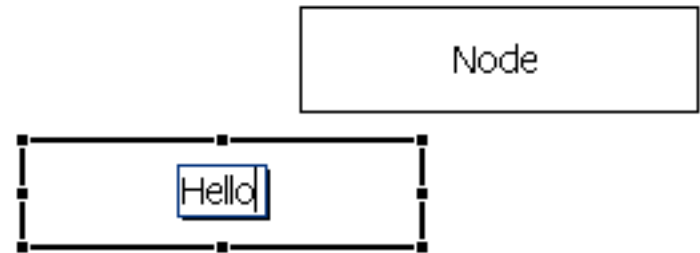
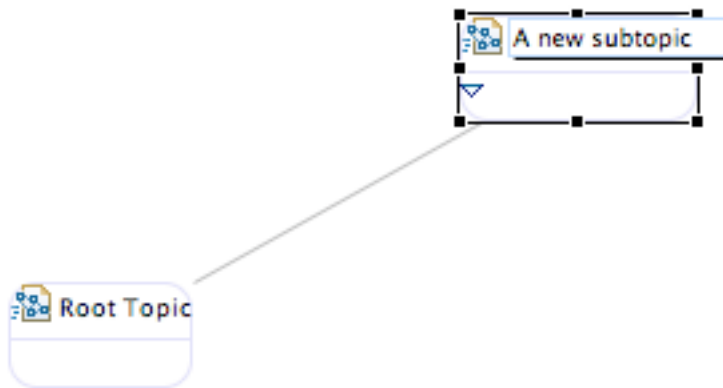
- **Megjeleníthető attribútum típusok**
 - **IPropertyDescriptor**
 - Általános tulajdonság leíró
 - Kötelező: ID, név
 - Opcionális: leírás, kategória, label provider (megjelenítéshez), cell editor (szerkesztéshez), context-sensitive help ID
 - **Kész implementációk:**
 - **PropertyDescriptor** (csak olvasható)
 - **TextPropertyDescriptor** (string attribútumok)
 - **CheckboxPropertyDescriptor** (boolean attribútumok)
 - **ComboBoxPropertyDescriptor** (enum attribútumok)
 - **ColorPropertyDescriptor** (szín attribútumok)

Modell tulajdonságok

- Tulajdonságok hozzárendelése
 - EditPart getAdapter()-ében visszaadunk egy IPropertySource példányt
 - Generikus IPropertySource implementáció (pl. EMF reflexív API-jával, ld. később)

Direct editing

- Cél: gyors szerkesztés közvetlenül a rajzterületen



Direct editing

- Szükséges:
 - DirectEditManager
 - Cell editor inicializálása
 - CellEditorLocator
 - Szerkesztő koordinátáinak meghatározása
 - Command
 - Modell manipuláció, undo/redo információ tárolása
 - Modell függő
 - DirectEditPolicy
 - DirectEditRequestből megfelelő Command előállítása
 - EditPart-ok felkészítése
 - DirectEditManager példányosítása és megjelenítése a Request hatására
 - DirectEditPolicy telepítése
 - Editor felkészítése
 - DirectEditAction regisztrálása

Direct Editing - DirectEditManager

- JFace-es cell editor feltöltése a megfelelő értékkel

```
public class TestDirectEditManager
    extends DirectEditManager {
    private ModelElement iModelElement;
    public TestDirectEditManager(
        GraphicalEditPart source,
        Class editorType,
        CellEditorLocator locator) {
        super(source, editorType, locator)
        iModelElement =
            (ModelElement) source.getModel();
    }
    protected void initCellEditor() {
        getCellEditor().setValue(iModelElement.getName());
        Text text = (Text) getCellEditor().getControl();
        text.selectAll();
    }
}
```

Direct Editing - CellEditorLocator

- Szerkesztő téglalap helyének számítása

```
public void relocate(CellEditor
cellEditor) {
    Text text = (Text)
cellEditor.getControl();
    Point pref = text.computeSize(-1, -1);
    Rectangle rect = null;
    rect = figure.getBounds().getCopy();
    figure.translateToAbsolute(rect);
    text.setBounds(rect.x, rect.y,
        rect.width, rect.height);
}
```

Direct Editing - DirectEditPolicy

- Command létrehozása Request alapján

```
protected Command getDirectEditCommand(  
    DirectEditRequest  
    request) {  
    Point p = request.getLocation();  
    SetNameCommand command =  
        new SetNameCommand();  
    command.setModel(getHost().getModel());  
    command.setText((String)  
        request.getCellEditor().getValue());  
    return command;  
}
```

Direct Editing – EditPartok felkészítése

■ DirectEditManager példányosítása Request után

```
private MyDirectEditManager directManager = null;
public void performRequest(Request req) {
    if (req.getType()
        .equals(RequestConstants.REQ_DIRECT_EDIT)) {
        performDirectEdit();
        return;
    }
    super.performRequest(req);
}
```

Direct Editing – EditPartok felkészítése

- DirectEditManager példányosítása Request után

```
private void performDirectEdit() {  
    if (directManager == null) {  
        directManager =  
            new MyDirectEditManager(this,  
                TextCellEditor.class,  
                new MyCellEditorLocator  
                    (getFigure()));  
    }  
    directManager.show();  
}
```

Direct Editing – EditPartok felkészítése

- DirectEditPolicy aktiválása

```
protected void createEditPolicies() {  
    installEditPolicy(  
        EditPolicy.DIRECT_EDIT_ROLE,  
        new MyDirectEditPolicy());  
}
```

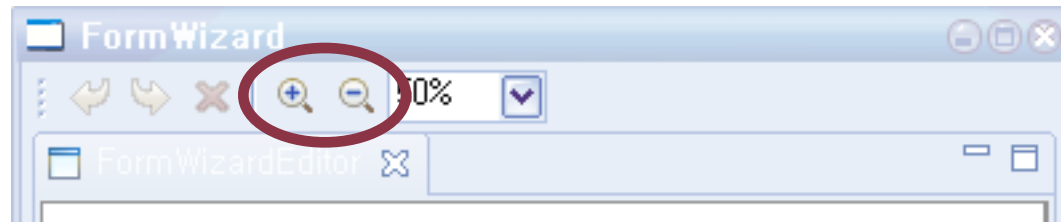
Direct Editing – Editor felkészítése

- DirectEditAction regisztrálása

```
protected void createAction() {  
    super.createAction();  
    IAction action = new DirectEditAction  
(  
  
(IWorkbenchPart) this);  
    registry.registerAction(action);  
    getSelectionActions().add  
(action.getId());  
}
```

Grafikus szerkesztő nagyítása

- Cél: diagram vektoros nagyítása (kicsinyítése)



Grafikus nézet nagyítása 2.

- Szükséges:
 - Editor felkészítése
 - ScalableRootEditPart a háttérbe
 - ZoomManager
 - ZoomInAction, ZoomOutAction
 - getAdapter() módosítása
 - ActionBarContributor felkészítése
 - ZoomInRetargetAction, ZoomOutRetargetAction

Grafikus nézet nagyítása 3.1

- Editor felkészítése

```
protected void configureGraphicalViewer() {  
    [...]  
    ScalableRootEditPart rootEditPart =  
        new ScalableRootEditPart();  
    viewer.setRootEditPart(rootEditPart);  
  
    ZoomManager manager =  
    rootEditPart.getZoomManager();  
    IAction action = new ZoomInAction(manager);  
    getActionRegistry().registerAction(action);  
    action = new ZoomOutAction(manager);  
    getActionRegistry().registerAction(action);  
    [...]  
}
```

Grafikus nézet nagyítása 3.2

■ Editor felkészítése

```
public Object getAdapter(Class type) {  
    if (type == ZoomManager.class)  
        return ( (ScalableRootEditPart)  
            getGraphicalViewer().getRootEditPart  
            ())  
            .getZoomManager();  
    return super.getAdapter(type);  
}
```

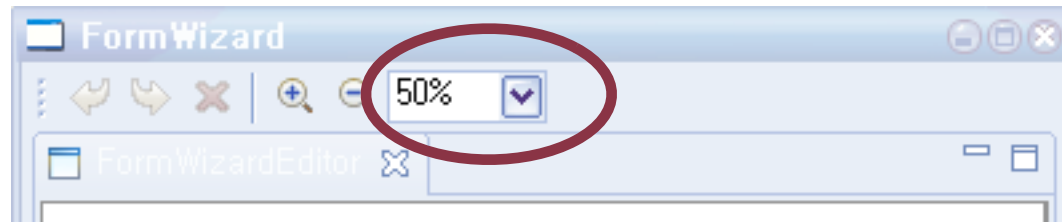
Grafikus nézet nagyítása 4.

■ ActionBarContributor felkészítése

```
protected void buildActions() {  
    [...]  
    addRetargetAction(new ZoomInRetargetAction());  
    addRetargetAction(new ZoomOutRetargetAction());  
}  
  
public void contributeToToolBar(  
    IToolBarManager toolBarManager) {  
    toolBarManager.add(new Separator());  
    toolBarManager.add(  
        getActionRegistry().getAction(  
            GEFActionConstants.ZOOM_IN));  
    toolBarManager.add(  
        getActionRegistry().getAction(  
            GEFActionConstants.ZOOM_OUT));  
}
```

Grafikus nézet nagyítása 5.

- Fix lehetőségek



Grafikus nézet nagyítása 6.

- **ActionBarContributor.contributeToToolBar()**

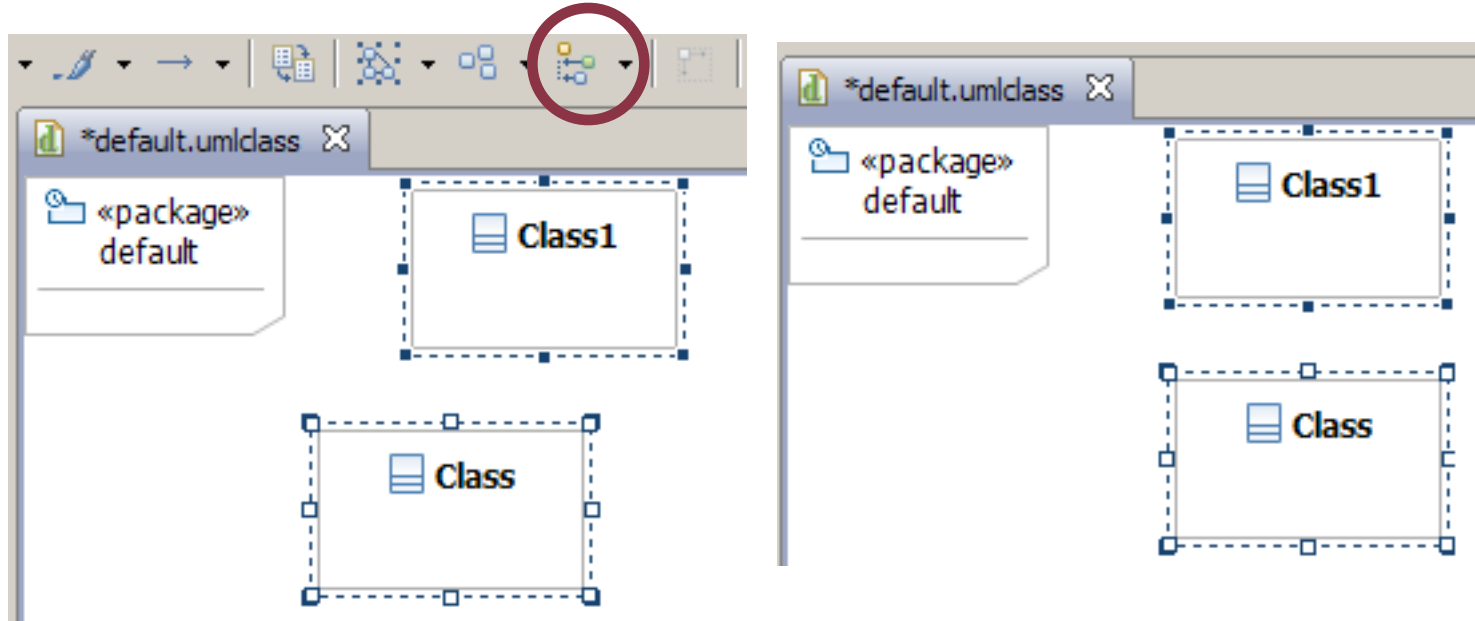
```
toolBarManager.add(  
    new ZoomComboContributionItem(getPage()));
```

- **Editor.configureGraphicalViewer()**

```
double[] zoomLevels = new double[]  
    { 0.25, 0.5, 0.75, 1.0, 1.5, 2.0, 2.5, 3.0, 10.0, 20.0 };  
manager.setZoomLevels(zoomLevels);  
ArrayList zoomContributions = new ArrayList();  
zoomContributions.add(ZoomManager.FIT_ALL);  
zoomContributions.add(ZoomManager.FIT_HEIGHT);  
zoomContributions.add(ZoomManager.FIT_WIDTH);  
manager.setZoomLevelContributions  
    (zoomContributions);
```

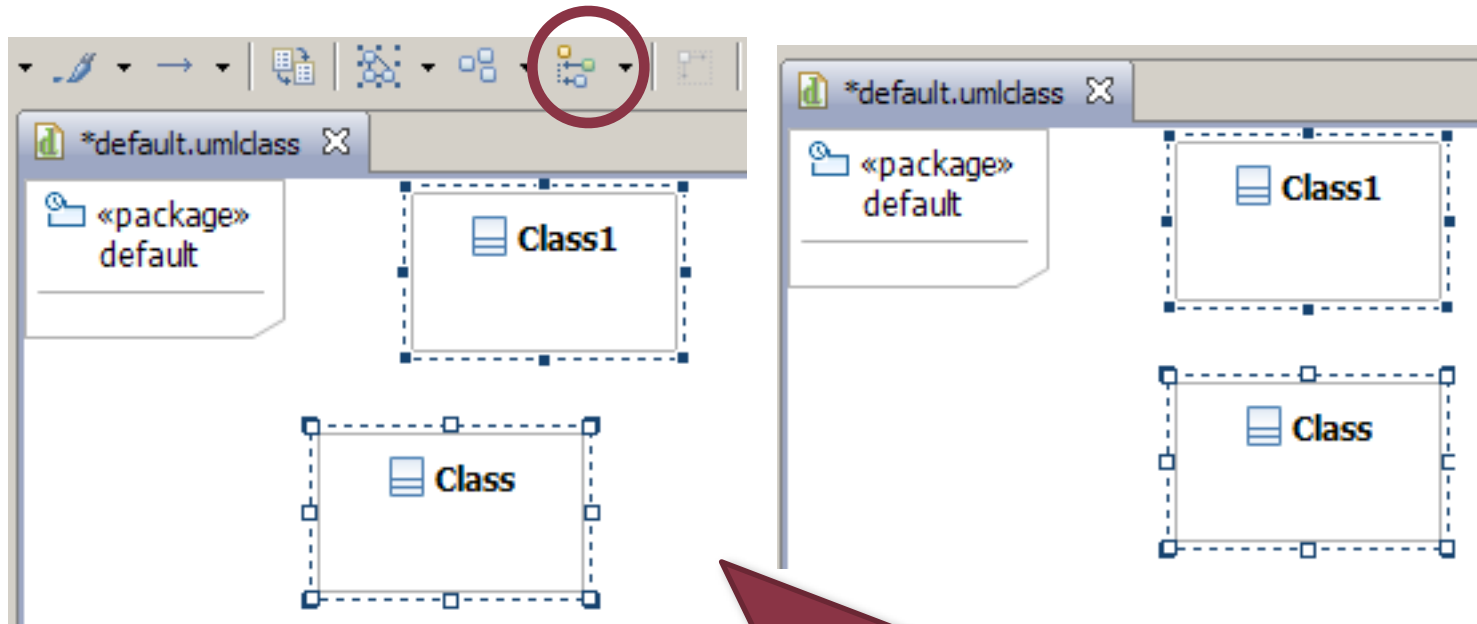
Igazítás

- Cél: alakzatok egymáshoz igazítása



Igazítás

- Cél: alakzatok egymáshoz igazítása

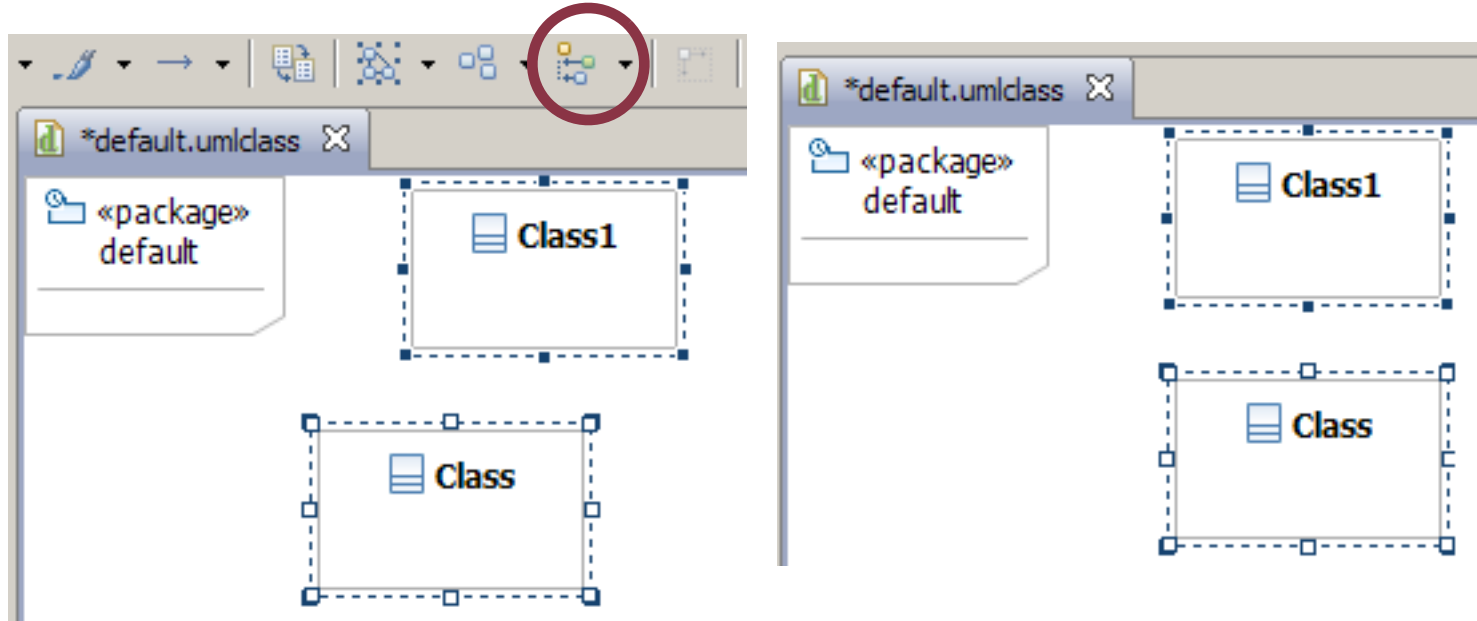


Használat:

1. Alakzatok kijelölés
2. Igazító gomb megnyomása

Igazítás

- Cél: alakzatok egymáshoz igazítása



Igazítás II.

- Szükséges:
 - Editor kiegészítése
 - 6 AlignmentAction regisztrálása
 - ActionBarContributor kiegészítése
 - 6 AlignmentRetargetAction hozzáadása

Igazítás III.

- **ActionBarContributor.contributeToToolBar()**

```
toolBarManager.add(getActionRegistry()  
    .getAction(GEFActionConstants.ALIGN_LEFT));  
// CENTER, RIGHT, TOP, MIDDLE, BOTTOM
```

- **ActionBarContributor.buildActions()**

```
addRetargetAction(new  
AlignmentRetargetAction  
    (PositionConstants.LEFT));  
// PositionConstants.CENTER  
// .RIGHT  
// .TOP  
// .MIDDLE  
// .BOTTOM
```

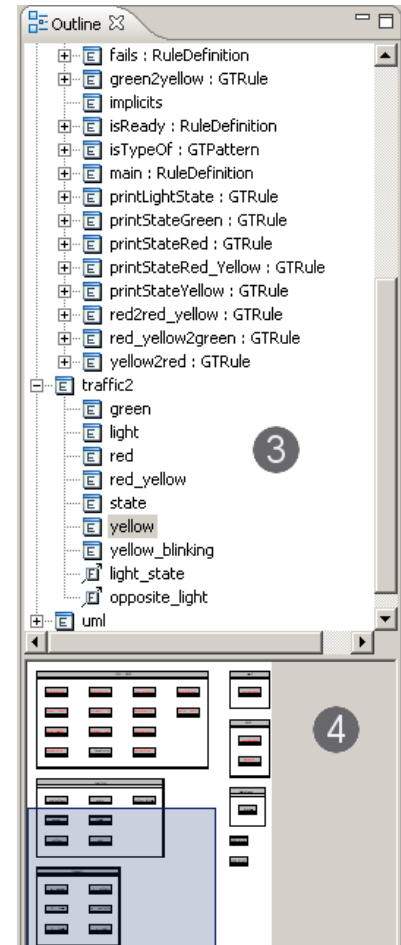
Igazítás IV.

- Editor.createActions()

```
public void createActions () {  
    [...]  
    IAction action =  
        new AlignmentAction  
        ((IWorkbenchPart) this,  
         PositionConstants.LEFT);  
    registry.registerAction(action);  
    getSelectionActions().add  
    (action.getId());  
    [...]  
}
```

Fa model nézet, kicsinyített vázlat

- GEF tartalmaz egy saját TreeViewer implementációt
 - TreeEditPart-okat jelenít meg
 - Fa nézetű szerkesztőhöz is felhasználható
 - Kiválasztás szinkronizálható a grafikus szerkesztőhöz
- Outline nézet alja: scrollozható thumbnail vázlat



Fa modellnézet

- Szükséges:
 - Saját ContentOutlinePage osztály
 - Editor belső osztályaként megvalósítva
 - TreeEditPart-ok a modellhez
 - IPropertySource támogatás
 - TreeEditPartFactory
 - TreeEditPolicy
 - Csak akkor, ha a fa nézetben kell szerkesztés
 - Saját SelectionSynchronizer
 - Csak akkor, ha a kijelölés szinkronizálandó

Fa modellnézet

- MyContentOutlinePage belső osztály

```
import org.eclipse.gef.ui.parts.ContentOutlinePage;
class MyContentOutlinePage extends
    ContentOutlinePage {
    private SashForm sash;
    public MyContentOutlinePage() {
        super(new TreeViewer());
    }
    public void createControl(Composite parent) {
        sash = new SashForm(parent, SWT.VERTICAL);
    }
    public Control getControl() {
        return sash;
    }
}
```

Fa modellnézet

- MyContentOutlinePage belső osztály

```
import org.eclipse.gef.ui.parts.ContentOutlinePage;
class MyContentOutlinePage extends
    ContentOutlinePage {
    private SashForm sash;
    public MyContentOutlinePage() {
        super(new TreeViewer());
    }
    public void createControl(Composite parent) {
        sash = new SashForm(parent, SWT.VERTICAL);
    }
    public Control getControl() {
        return sash;
    }
}
```

SWT konténer,
két elválasztott
részből

Fa modellnézet

- **MyContentOutlinePage** belső osztály “bekötése”

```
public Object getAdapter(Class type) {  
    if (type == IContentOutlinePage.class)  
    {  
        return new MyContentOutlinePage();  
    }  
    return super.getAdapter(type);  
}
```

Fa modellnézet

- TreeEditart-ok – Ősosztály

```
public abstract class MyTreeEditPart
    extends AbstractTreeEditPart
    implements PropertyChangeListener {
    public void activate() {
        super.activate();
        ((AbstractModel)getModel())
            .addPropertyChangeListener(this);
    }
    public void deactivate() {
        ((AbstractModel)getModel())
            .removePropertyChangeListener(this);
        super.deactivate();
    }
}
```

Fa modellnézet

- TreeEditart-ok – Ősosztály

```
public abstract class MyTreeEditPart
    extends AbstractTreeEditPart
    implements PropertyChangeListener {
    public void activate() {
        super.activate();
        ((AbstractModel)getModel())
            .addPropertyChangeListener(this);
    }
    public void deactivate() {
        ((AbstractModel)getModel())
            .removePropertyChangeListener(this);
        super.deactivate();
    }
}
```

Értesítési
mechanizmus
bekötése

Fa modellnézet

- TreeEditPart-ok – gyökérelem (nem látszik)

```
public abstract class ContentsTreeEditPart
    extends MyTreeEditPart {
    [...]
    protected List getModelChildren() {
        return ((ContentsModel)getModel
        ()).getChildren();
    }
    public void propertyChange(PropertyChangeEvent
    event) {
        if(event.getPropertyName()
        .equals(ContentsModel.P_CHILDREN))
        refreshChildren();
    }
}
```

Fa modellnézet

- TreeEditPart-ok – gyerekelemek

```
public abstract class ChildTreeEditPart
    extends MyTreeEditPart {
    [...]
    protected void refreshVisuals() {
        ChildModel model = (ChildModel) getModel();
        setWidgetText(model.getText());
    }
    public void propertyChange(PropertyChangeEvent
event) {
        if (event.getPropertyName()
            .equals(ChildModel.P_TEXT))
            refreshVisuals();
    }
}
```

Fa modellnézet

- **TreeEditPartFactory**

```
public class TreeEditPartFactory
    implements EditPartFactory {
    public EditPart createEditPart
        (EditPart context, Object model) {
        EditPart part = null;
        if (model instanceof ContentsModel)
            part = new ContentsTreeEditPart();
        else if (model instanceof ChildModel)
            part = new ChildTreeEditPart();
        if (part != null)
            part.setModel(model);
        return part;
    }
}
```

Fa modellnézet

- **TreeViewer és EditPartok összekapcsolása**

```
public class MyContentOutlinePage
    extends ContentOutlinePage {
    [...]
    public void createControl(Composite parent) {
        getViewer().createControl(sash);
        getViewer().setEditDomain(getEditDomain());
        getViewer().setEditPartFactory(
            new TreeEditPartFactory());
        getViewer().setContents(contentsModel);
        getSelectionSynchronizer()
            .addViewer(getViewer());
    }
}
```

Fa modellnézet

- TreeViewer és EditPartok összekapcsolása

```
public class MyContentOutlinePage
    extends ContentOutlinePage {
    [...]
    public void createControl(Composite parent) {
        getViewer().createControl(sash);
        getViewer().setEditDomain(getEditDomain());
        getViewer().setEditPartFactory(
            new TreeEditPartFactory());
        getViewer().setContents(contentsModel);
        getSelectionSynchronizer()
            .addViewer(getViewer());
    }
}
```

Külső editor
osztály private
függvénye

Fa modellnézet

- **TreeViewer és EditPartok összekapcsolása**

```
public class MyContentOutlinePage
    extends ContentOutlinePage {
    [...]
    public void createControl(Composite parent) {
        getViewer().createControl(sash);
        getViewer().setEditDomain(getEditDomain());
        getViewer().setEditPartFactory(
            new TreeEditPartFactory());
        getViewer().setContents(contentsModel);
        getSelectionSynchronizer()
            .addViewer(getViewer());
    }
}
```

Fa modellnézet

- **TreeViewer és EditPartok összekapcsolása**

```
public class MyContentOutlinePage
    extends ContentOutlinePage {
    [...]
    public void dispose() {
        getSelectionSynchronizer()
            .removeViewer(getViewer());
        super.dispose();
    }
}
```

■ TreeEditPolicy

- COMPONENT_ROLE: ComponentEditPolicy
 - createDeleteCommand()
- TREE_CONTAINER_ROLE: TreeContainerEditPolicy
 - getAddCommand()
 - Tree-n belüli drag and drop
 - getCreateCommand()
 - Konténeren belül jön létre a gyerek elem
 - getMoveChildrenCommand()
 - Konténeren belüli mozgatás

Kicsinyített vázlat

- Thumbnail összerakása

```
public class MyContentOutlinePage
    extends ContentOutlinePage {
    [...]
    public void createControl() {
        [...]
        Canvas canvas = new Canvas(sash, SWT.BORDER);
        LightweightSystem lws =
            new LightweightSystem(canvas);
        thumbnail = new ScrollableThumbnail(
            (Viewport) ((ScalableRootEditPart)
                getGraphicalViewer()
                    .getRootEditPart()).getFigure());
```

Kicsinyített vázlat

- Thumbnail összerakása

```
public class MyContentOutlinePage
    extends ContentOutlinePage {
    [...]
    public void dispose() {
        getSelectionSynchronizer()
            .removeViewer(getViewer());

        if (getGraphicalViewer().getControl() != null
            && !getGraphicalViewer().getControl
            ().isDisposed())
            getGraphicalViewer().getControl()
                .removeDisposeListener(disposeListener);
        super.dispose();
        iOutlinePage = null;
    }
}
```

Fa modellnézet, kicsinyített vázlat

- A kész Outline nézet
 - Zoom-ot követi
 - “Lusta” renderelés

