Министерство образования и науки Украины

Национальный технический университет Украины „КПИ”

Факультет информатики и вычислительной техники

Кафедра вычислительной техники

Автоматизация проектирования компьютерных систем

Лабораторная работа №1

Редактор блок-схем алгоритмов.

Выполнил

ст. гр. ИВ-82

Олещук О.А.

Киев

2011

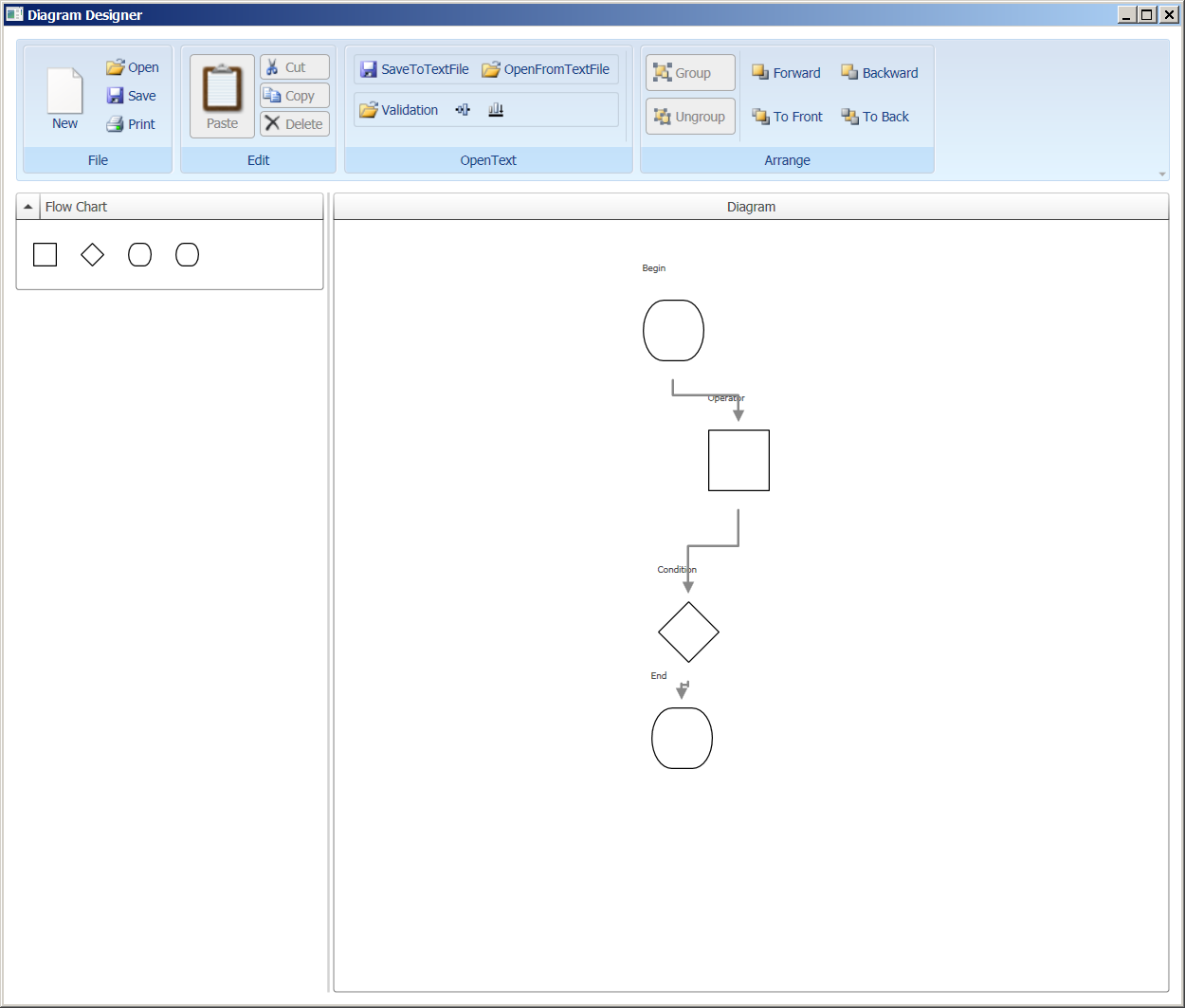
Цель: получение навыков построения редактора блок-схем алгоритмов. Разработка интерфейса пользователя и функционального наполнения. Разработка средств преобразования форматов сохранения данных.

Задание:

Тип редактора-редактор графических схем алгоритмов (ГСА)

Тип формата-текстовый

Скриншот программы:



Листинг программы:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

namespace DiagramDesigner

{

//These attributes identify the types of the named parts that are used for templating

[TemplatePart(Name = "PART\_DragThumb", Type = typeof(DragThumb))]

[TemplatePart(Name = "PART\_ResizeDecorator", Type = typeof(Control))]

[TemplatePart(Name = "PART\_ConnectorDecorator", Type = typeof(Control))]

[TemplatePart(Name = "PART\_ContentPresenter", Type = typeof(ContentPresenter))]

[TemplatePart(Name = "PART\_BlockContentPresenter", Type = typeof(TextBlock))]

public class DesignerItem : ContentControl, ISelectable, IGroupable

{

public static RoutedCommand Properties = new RoutedCommand();

#region NodeType

public NodeTypes NodeType

{

get { return (NodeTypes)GetValue(NodeTypeProperty); }

set { SetValue(NodeTypeProperty, value); }

}

public static readonly DependencyProperty NodeTypeProperty =

DependencyProperty.Register("NodeType", typeof(NodeTypes), typeof(DesignerItem));

#endregion

#region BlockContent

private string blockContent;

public string BlockContent

{

get

{

return this.blockContent;

}

set

{

this.blockContent = value;

}

}

#endregion

public bool ConteinsSignal(string signal)

{

string[] signals = this.BlockContent.Split(',');

foreach (string s in signals)

if (s == signal) return true;

return false;

}

#region ID

private Guid id;

public Guid ID

{

get { return id; }

}

#endregion

#region ParentID

public Guid ParentID

{

get { return (Guid)GetValue(ParentIDProperty); }

set { SetValue(ParentIDProperty, value); }

}

public static readonly DependencyProperty ParentIDProperty = DependencyProperty.Register("ParentID", typeof(Guid), typeof(DesignerItem));

#endregion

#region IsGroup

public bool IsGroup

{

get { return (bool)GetValue(IsGroupProperty); }

set { SetValue(IsGroupProperty, value); }

}

public static readonly DependencyProperty IsGroupProperty =

DependencyProperty.Register("IsGroup", typeof(bool), typeof(DesignerItem));

#endregion

#region IsSelected Property

public bool IsSelected

{

get { return (bool)GetValue(IsSelectedProperty); }

set { SetValue(IsSelectedProperty, value); }

}

public static readonly DependencyProperty IsSelectedProperty =

DependencyProperty.Register("IsSelected",

typeof(bool),

typeof(DesignerItem),

new FrameworkPropertyMetadata(false));

#endregion

#region DragThumbTemplate Property

// can be used to replace the default template for the DragThumb

public static readonly DependencyProperty DragThumbTemplateProperty =

DependencyProperty.RegisterAttached("DragThumbTemplate", typeof(ControlTemplate), typeof(DesignerItem));

public static ControlTemplate GetDragThumbTemplate(UIElement element)

{

return (ControlTemplate)element.GetValue(DragThumbTemplateProperty);

}

public static void SetDragThumbTemplate(UIElement element, ControlTemplate value)

{

element.SetValue(DragThumbTemplateProperty, value);

}

#endregion

#region ConnectorDecoratorTemplate Property

// can be used to replace the default template for the ConnectorDecorator

public static readonly DependencyProperty ConnectorDecoratorTemplateProperty =

DependencyProperty.RegisterAttached("ConnectorDecoratorTemplate", typeof(ControlTemplate), typeof(DesignerItem));

public static ControlTemplate GetConnectorDecoratorTemplate(UIElement element)

{

return (ControlTemplate)element.GetValue(ConnectorDecoratorTemplateProperty);

}

public static void SetConnectorDecoratorTemplate(UIElement element, ControlTemplate value)

{

element.SetValue(ConnectorDecoratorTemplateProperty, value);

}

#endregion

#region IsDragConnectionOver

// while drag connection procedure is ongoing and the mouse moves over

// this item this value is true; if true the ConnectorDecorator is triggered

// to be visible, see template

public bool IsDragConnectionOver

{

get { return (bool)GetValue(IsDragConnectionOverProperty); }

set { SetValue(IsDragConnectionOverProperty, value); }

}

public static readonly DependencyProperty IsDragConnectionOverProperty =

DependencyProperty.Register("IsDragConnectionOver",

typeof(bool),

typeof(DesignerItem),

new FrameworkPropertyMetadata(false));

#endregion

static DesignerItem()

{

// set the key to reference the style for this control

FrameworkElement.DefaultStyleKeyProperty.OverrideMetadata(

typeof(DesignerItem), new FrameworkPropertyMetadata(typeof(DesignerItem)));

}

private void Properties\_Executed(object sender, ExecutedRoutedEventArgs e)

{

DIPropWin diPW = new DIPropWin(this);

diPW.ShowDialog();

}

public DesignerItem(Guid id)

{

this.id = id;

this.Loaded += new RoutedEventHandler(DesignerItem\_Loaded);

this.CommandBindings.Add(new CommandBinding(DesignerItem.Properties, Properties\_Executed));

}

public DesignerItem()

: this(Guid.NewGuid())

{

}

protected override void OnPreviewMouseDown(MouseButtonEventArgs e)

{

base.OnPreviewMouseDown(e);

DesignerCanvas designer = VisualTreeHelper.GetParent(this) as DesignerCanvas;

// update selection

if (designer != null)

{

if ((Keyboard.Modifiers & (ModifierKeys.Shift | ModifierKeys.Control)) != ModifierKeys.None)

if (this.IsSelected)

{

designer.SelectionService.RemoveFromSelection(this);

}

else

{

designer.SelectionService.AddToSelection(this);

}

else if (!this.IsSelected)

{

designer.SelectionService.SelectItem(this);

}

Focus();

}

e.Handled = false;

}

void DesignerItem\_Loaded(object sender, RoutedEventArgs e)

{

if (base.Template != null)

{

ContentPresenter contentPresenter =

this.Template.FindName("PART\_ContentPresenter", this) as ContentPresenter;

if (contentPresenter != null)

{

UIElement contentVisual = VisualTreeHelper.GetChild(contentPresenter, 0) as UIElement;

if (contentVisual != null)

{

DragThumb thumb = this.Template.FindName("PART\_DragThumb", this) as DragThumb;

if (thumb != null)

{

ControlTemplate template =

DesignerItem.GetDragThumbTemplate(contentVisual) as ControlTemplate;

if (template != null)

thumb.Template = template;

}

}

}

}

}

}

public enum NodeTypes

{

Begin,Operator,Condition,End

}

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.Globalization;

using System.IO;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Markup;

using System.Windows.Media;

using System.Xml;

using System.Xml.Linq;

using Microsoft.Win32;

using System.Text;

namespace DiagramDesigner

{

public partial class DesignerCanvas

{

public static RoutedCommand Group = new RoutedCommand();

public static RoutedCommand Ungroup = new RoutedCommand();

public static RoutedCommand BringForward = new RoutedCommand();

public static RoutedCommand BringToFront = new RoutedCommand();

public static RoutedCommand SendBackward = new RoutedCommand();

public static RoutedCommand SendToBack = new RoutedCommand();

public static RoutedCommand AlignTop = new RoutedCommand();

public static RoutedCommand AlignVerticalCenters = new RoutedCommand();

public static RoutedCommand AlignBottom = new RoutedCommand();

public static RoutedCommand AlignLeft = new RoutedCommand();

public static RoutedCommand AlignHorizontalCenters = new RoutedCommand();

public static RoutedCommand AlignRight = new RoutedCommand();

public static RoutedCommand DistributeHorizontal = new RoutedCommand();

public static RoutedCommand DistributeVertical = new RoutedCommand();

public static RoutedCommand SelectAll = new RoutedCommand();

public static RoutedCommand SeveToTextFile = new RoutedCommand();

public static RoutedCommand OpenFromTextFile = new RoutedCommand();

public static RoutedCommand Validation= new RoutedCommand();

public DesignerCanvas()

{

this.CommandBindings.Add(new CommandBinding(ApplicationCommands.New, New\_Executed));

this.CommandBindings.Add(new CommandBinding(ApplicationCommands.Open, Open\_Executed));

this.CommandBindings.Add(new CommandBinding(ApplicationCommands.Save, Save\_Executed));

this.CommandBindings.Add(new CommandBinding(ApplicationCommands.Print, Print\_Executed));

this.CommandBindings.Add(new CommandBinding(ApplicationCommands.Cut, Cut\_Executed, Cut\_Enabled));

this.CommandBindings.Add(new CommandBinding(ApplicationCommands.Copy, Copy\_Executed, Copy\_Enabled));

this.CommandBindings.Add(new CommandBinding(ApplicationCommands.Paste, Paste\_Executed, Paste\_Enabled));

this.CommandBindings.Add(new CommandBinding(ApplicationCommands.Delete, Delete\_Executed, Delete\_Enabled));

this.CommandBindings.Add(new CommandBinding(DesignerCanvas.Group, Group\_Executed, Group\_Enabled));

this.CommandBindings.Add(new CommandBinding(DesignerCanvas.Ungroup, Ungroup\_Executed, Ungroup\_Enabled));

this.CommandBindings.Add(new CommandBinding(DesignerCanvas.BringForward, BringForward\_Executed, Order\_Enabled));

this.CommandBindings.Add(new CommandBinding(DesignerCanvas.BringToFront, BringToFront\_Executed, Order\_Enabled));

this.CommandBindings.Add(new CommandBinding(DesignerCanvas.SendBackward, SendBackward\_Executed, Order\_Enabled));

this.CommandBindings.Add(new CommandBinding(DesignerCanvas.SendToBack, SendToBack\_Executed, Order\_Enabled));

this.CommandBindings.Add(new CommandBinding(DesignerCanvas.AlignTop, AlignTop\_Executed, Align\_Enabled));

this.CommandBindings.Add(new CommandBinding(DesignerCanvas.AlignVerticalCenters, AlignVerticalCenters\_Executed, Align\_Enabled));

this.CommandBindings.Add(new CommandBinding(DesignerCanvas.AlignBottom, AlignBottom\_Executed, Align\_Enabled));

this.CommandBindings.Add(new CommandBinding(DesignerCanvas.AlignLeft, AlignLeft\_Executed, Align\_Enabled));

this.CommandBindings.Add(new CommandBinding(DesignerCanvas.AlignHorizontalCenters, AlignHorizontalCenters\_Executed, Align\_Enabled));

this.CommandBindings.Add(new CommandBinding(DesignerCanvas.AlignRight, AlignRight\_Executed, Align\_Enabled));

this.CommandBindings.Add(new CommandBinding(DesignerCanvas.DistributeHorizontal, DistributeHorizontal\_Executed, Distribute\_Enabled));

this.CommandBindings.Add(new CommandBinding(DesignerCanvas.DistributeVertical, DistributeVertical\_Executed, Distribute\_Enabled));

this.CommandBindings.Add(new CommandBinding(DesignerCanvas.SelectAll, SelectAll\_Executed));

this.CommandBindings.Add(new CommandBinding(DesignerCanvas.SeveToTextFile, SeveToTextFile\_Executed));

this.CommandBindings.Add(new CommandBinding(DesignerCanvas.OpenFromTextFile, OpenFromTextFile\_Executed));

this.CommandBindings.Add(new CommandBinding(DesignerCanvas.Validation, Validation\_Executed));

SelectAll.InputGestures.Add(new KeyGesture(Key.A, ModifierKeys.Control));

this.AllowDrop = true;

Clipboard.Clear();

}

#region New Command

private void New\_Executed(object sender, ExecutedRoutedEventArgs e)

{

this.Children.Clear();

this.SelectionService.ClearSelection();

}

#endregion

#region Open Command

private void Open\_Executed(object sender, ExecutedRoutedEventArgs e)

{

XElement root = LoadSerializedDataFromFile();

if (root == null)

return;

this.Children.Clear();

this.SelectionService.ClearSelection();

IEnumerable<XElement> itemsXML = root.Elements("DesignerItems").Elements("DesignerItem");

foreach (XElement itemXML in itemsXML)

{

Guid id = new Guid(itemXML.Element("ID").Value);

DesignerItem item = DeserializeDesignerItem(itemXML, id, 0, 0);

this.Children.Add(item);

SetConnectorDecoratorTemplate(item);

}

this.InvalidateVisual();

IEnumerable<XElement> connectionsXML = root.Elements("Connections").Elements("Connection");

foreach (XElement connectionXML in connectionsXML)

{

Guid sourceID = new Guid(connectionXML.Element("SourceID").Value);

Guid sinkID = new Guid(connectionXML.Element("SinkID").Value);

String sourceConnectorName = connectionXML.Element("SourceConnectorName").Value;

String sinkConnectorName = connectionXML.Element("SinkConnectorName").Value;

Connector sourceConnector = GetConnector(sourceID, sourceConnectorName);

Connector sinkConnector = GetConnector(sinkID, sinkConnectorName);

Connection connection = new Connection(sourceConnector, sinkConnector);

Canvas.SetZIndex(connection, Int32.Parse(connectionXML.Element("zIndex").Value));

this.Children.Add(connection);

}

}

#endregion

#region Save Command

private void Save\_Executed(object sender, ExecutedRoutedEventArgs e)

{

IEnumerable<DesignerItem> designerItems = this.Children.OfType<DesignerItem>();

IEnumerable<Connection> connections = this.Children.OfType<Connection>();

XElement designerItemsXML = SerializeDesignerItems(designerItems);

XElement connectionsXML = SerializeConnections(connections);

XElement root = new XElement("Root");

root.Add(designerItemsXML);

root.Add(connectionsXML);

SaveFile(root);

}

#endregion

#region Print Command

private void Print\_Executed(object sender, ExecutedRoutedEventArgs e)

{

SelectionService.ClearSelection();

PrintDialog printDialog = new PrintDialog();

if (true == printDialog.ShowDialog())

{

printDialog.PrintVisual(this, "WPF Diagram");

}

}

#endregion

#region Copy Command

private void Copy\_Executed(object sender, ExecutedRoutedEventArgs e)

{

CopyCurrentSelection();

}

private void Copy\_Enabled(object sender, CanExecuteRoutedEventArgs e)

{

e.CanExecute = SelectionService.CurrentSelection.Count() > 0;

}

#endregion

#region Paste Command

private void Paste\_Executed(object sender, ExecutedRoutedEventArgs e)

{

XElement root = LoadSerializedDataFromClipBoard();

if (root == null)

return;

// create DesignerItems

Dictionary<Guid, Guid> mappingOldToNewIDs = new Dictionary<Guid, Guid>();

List<ISelectable> newItems = new List<ISelectable>();

IEnumerable<XElement> itemsXML = root.Elements("DesignerItems").Elements("DesignerItem");

double offsetX = Double.Parse(root.Attribute("OffsetX").Value, CultureInfo.InvariantCulture);

double offsetY = Double.Parse(root.Attribute("OffsetY").Value, CultureInfo.InvariantCulture);

foreach (XElement itemXML in itemsXML)

{

Guid oldID = new Guid(itemXML.Element("ID").Value);

Guid newID = Guid.NewGuid();

mappingOldToNewIDs.Add(oldID, newID);

DesignerItem item = DeserializeDesignerItem(itemXML, newID, offsetX, offsetY);

this.Children.Add(item);

SetConnectorDecoratorTemplate(item);

newItems.Add(item);

}

// update group hierarchy

SelectionService.ClearSelection();

foreach (DesignerItem el in newItems)

{

if (el.ParentID != Guid.Empty)

el.ParentID = mappingOldToNewIDs[el.ParentID];

}

foreach (DesignerItem item in newItems)

{

if (item.ParentID == Guid.Empty)

{

SelectionService.AddToSelection(item);

}

}

// create Connections

IEnumerable<XElement> connectionsXML = root.Elements("Connections").Elements("Connection");

foreach (XElement connectionXML in connectionsXML)

{

Guid oldSourceID = new Guid(connectionXML.Element("SourceID").Value);

Guid oldSinkID = new Guid(connectionXML.Element("SinkID").Value);

if (mappingOldToNewIDs.ContainsKey(oldSourceID) && mappingOldToNewIDs.ContainsKey(oldSinkID))

{

Guid newSourceID = mappingOldToNewIDs[oldSourceID];

Guid newSinkID = mappingOldToNewIDs[oldSinkID];

String sourceConnectorName = connectionXML.Element("SourceConnectorName").Value;

String sinkConnectorName = connectionXML.Element("SinkConnectorName").Value;

Connector sourceConnector = GetConnector(newSourceID, sourceConnectorName);

Connector sinkConnector = GetConnector(newSinkID, sinkConnectorName);

Connection connection = new Connection(sourceConnector, sinkConnector);

Canvas.SetZIndex(connection, Int32.Parse(connectionXML.Element("zIndex").Value));

this.Children.Add(connection);

SelectionService.AddToSelection(connection);

}

}

DesignerCanvas.BringToFront.Execute(null, this);

// update paste offset

root.Attribute("OffsetX").Value = (offsetX + 10).ToString();

root.Attribute("OffsetY").Value = (offsetY + 10).ToString();

Clipboard.Clear();

Clipboard.SetData(DataFormats.Xaml, root);

}

private void Paste\_Enabled(object sender, CanExecuteRoutedEventArgs e)

{

e.CanExecute = Clipboard.ContainsData(DataFormats.Xaml);

}

#endregion

#region Delete Command

private void Delete\_Executed(object sender, ExecutedRoutedEventArgs e)

{

DeleteCurrentSelection();

}

private void Delete\_Enabled(object sender, CanExecuteRoutedEventArgs e)

{

e.CanExecute = this.SelectionService.CurrentSelection.Count() > 0;

}

#endregion

#region Cut Command

private void Cut\_Executed(object sender, ExecutedRoutedEventArgs e)

{

CopyCurrentSelection();

DeleteCurrentSelection();

}

private void Cut\_Enabled(object sender, CanExecuteRoutedEventArgs e)

{

e.CanExecute = this.SelectionService.CurrentSelection.Count() > 0;

}

#endregion

#region Group Command

private void Group\_Executed(object sender, ExecutedRoutedEventArgs e)

{

var items = from item in this.SelectionService.CurrentSelection.OfType<DesignerItem>()

where item.ParentID == Guid.Empty

select item;

Rect rect = GetBoundingRectangle(items);

DesignerItem groupItem = new DesignerItem();

groupItem.IsGroup = true;

groupItem.Width = rect.Width;

groupItem.Height = rect.Height;

Canvas.SetLeft(groupItem, rect.Left);

Canvas.SetTop(groupItem, rect.Top);

Canvas groupCanvas = new Canvas();

groupItem.Content = groupCanvas;

Canvas.SetZIndex(groupItem, this.Children.Count);

this.Children.Add(groupItem);

foreach (DesignerItem item in items)

item.ParentID = groupItem.ID;

this.SelectionService.SelectItem(groupItem);

}

private void Group\_Enabled(object sender, CanExecuteRoutedEventArgs e)

{

int count = (from item in SelectionService.CurrentSelection.OfType<DesignerItem>()

where item.ParentID == Guid.Empty

select item).Count();

e.CanExecute = count > 1;

}

#endregion

#region Ungroup Command

private void Ungroup\_Executed(object sender, ExecutedRoutedEventArgs e)

{

var groups = (from item in SelectionService.CurrentSelection.OfType<DesignerItem>()

where item.IsGroup && item.ParentID == Guid.Empty

select item).ToArray();

foreach (DesignerItem groupRoot in groups)

{

var children = from child in SelectionService.CurrentSelection.OfType<DesignerItem>()

where child.ParentID == groupRoot.ID

select child;

foreach (DesignerItem child in children)

child.ParentID = Guid.Empty;

this.SelectionService.RemoveFromSelection(groupRoot);

this.Children.Remove(groupRoot);

UpdateZIndex();

}

}

private void Ungroup\_Enabled(object sender, CanExecuteRoutedEventArgs e)

{

var groupedItem = from item in SelectionService.CurrentSelection.OfType<DesignerItem>()

where item.ParentID != Guid.Empty

select item;

e.CanExecute = groupedItem.Count() > 0;

}

#endregion

#region BringForward Command

private void BringForward\_Executed(object sender, ExecutedRoutedEventArgs e)

{

List<UIElement> ordered = (from item in SelectionService.CurrentSelection

orderby Canvas.GetZIndex(item as UIElement) descending

select item as UIElement).ToList();

int count = this.Children.Count;

for (int i = 0; i < ordered.Count; i++)

{

int currentIndex = Canvas.GetZIndex(ordered[i]);

int newIndex = Math.Min(count - 1 - i, currentIndex + 1);

if (currentIndex != newIndex)

{

Canvas.SetZIndex(ordered[i], newIndex);

IEnumerable<UIElement> it = this.Children.OfType<UIElement>().Where(item => Canvas.GetZIndex(item) == newIndex);

foreach (UIElement elm in it)

{

if (elm != ordered[i])

{

Canvas.SetZIndex(elm, currentIndex);

break;

}

}

}

}

}

private void Order\_Enabled(object sender, CanExecuteRoutedEventArgs e)

{

//e.CanExecute = SelectionService.CurrentSelection.Count() > 0;

e.CanExecute = true;

}

#endregion

#region BringToFront Command

private void BringToFront\_Executed(object sender, ExecutedRoutedEventArgs e)

{

List<UIElement> selectionSorted = (from item in SelectionService.CurrentSelection

orderby Canvas.GetZIndex(item as UIElement) ascending

select item as UIElement).ToList();

List<UIElement> childrenSorted = (from UIElement item in this.Children

orderby Canvas.GetZIndex(item as UIElement) ascending

select item as UIElement).ToList();

int i = 0;

int j = 0;

foreach (UIElement item in childrenSorted)

{

if (selectionSorted.Contains(item))

{

int idx = Canvas.GetZIndex(item);

Canvas.SetZIndex(item, childrenSorted.Count - selectionSorted.Count + j++);

}

else

{

Canvas.SetZIndex(item, i++);

}

}

}

#endregion

#region SendBackward Command

private void SendBackward\_Executed(object sender, ExecutedRoutedEventArgs e)

{

List<UIElement> ordered = (from item in SelectionService.CurrentSelection

orderby Canvas.GetZIndex(item as UIElement) ascending

select item as UIElement).ToList();

int count = this.Children.Count;

for (int i = 0; i < ordered.Count; i++)

{

int currentIndex = Canvas.GetZIndex(ordered[i]);

int newIndex = Math.Max(i, currentIndex - 1);

if (currentIndex != newIndex)

{

Canvas.SetZIndex(ordered[i], newIndex);

IEnumerable<UIElement> it = this.Children.OfType<UIElement>().Where(item => Canvas.GetZIndex(item) == newIndex);

foreach (UIElement elm in it)

{

if (elm != ordered[i])

{

Canvas.SetZIndex(elm, currentIndex);

break;

}

}

}

}

}

#endregion

#region SendToBack Command

private void SendToBack\_Executed(object sender, ExecutedRoutedEventArgs e)

{

List<UIElement> selectionSorted = (from item in SelectionService.CurrentSelection

orderby Canvas.GetZIndex(item as UIElement) ascending

select item as UIElement).ToList();

List<UIElement> childrenSorted = (from UIElement item in this.Children

orderby Canvas.GetZIndex(item as UIElement) ascending

select item as UIElement).ToList();

int i = 0;

int j = 0;

foreach (UIElement item in childrenSorted)

{

if (selectionSorted.Contains(item))

{

int idx = Canvas.GetZIndex(item);

Canvas.SetZIndex(item, j++);

}

else

{

Canvas.SetZIndex(item, selectionSorted.Count + i++);

}

}

}

#endregion

#region AlignTop Command

private void AlignTop\_Executed(object sender, ExecutedRoutedEventArgs e)

{

var selectedItems = from item in SelectionService.CurrentSelection.OfType<DesignerItem>()

where item.ParentID == Guid.Empty

select item;

if (selectedItems.Count() > 1)

{

double top = Canvas.GetTop(selectedItems.First());

foreach (DesignerItem item in selectedItems)

{

double delta = top - Canvas.GetTop(item);

foreach (DesignerItem di in SelectionService.GetGroupMembers(item))

{

Canvas.SetTop(di, Canvas.GetTop(di) + delta);

}

}

}

}

private void Align\_Enabled(object sender, CanExecuteRoutedEventArgs e)

{

//var groupedItem = from item in SelectionService.CurrentSelection.OfType<DesignerItem>()

// where item.ParentID == Guid.Empty

// select item;

//e.CanExecute = groupedItem.Count() > 1;

e.CanExecute = true;

}

#endregion

#region AlignVerticalCenters Command

private void AlignVerticalCenters\_Executed(object sender, ExecutedRoutedEventArgs e)

{

var selectedItems = from item in SelectionService.CurrentSelection.OfType<DesignerItem>()

where item.ParentID == Guid.Empty

select item;

if (selectedItems.Count() > 1)

{

double bottom = Canvas.GetTop(selectedItems.First()) + selectedItems.First().Height / 2;

foreach (DesignerItem item in selectedItems)

{

double delta = bottom - (Canvas.GetTop(item) + item.Height / 2);

foreach (DesignerItem di in SelectionService.GetGroupMembers(item))

{

Canvas.SetTop(di, Canvas.GetTop(di) + delta);

}

}

}

}

#endregion

#region AlignBottom Command

private void AlignBottom\_Executed(object sender, ExecutedRoutedEventArgs e)

{

var selectedItems = from item in SelectionService.CurrentSelection.OfType<DesignerItem>()

where item.ParentID == Guid.Empty

select item;

if (selectedItems.Count() > 1)

{

double bottom = Canvas.GetTop(selectedItems.First()) + selectedItems.First().Height;

foreach (DesignerItem item in selectedItems)

{

double delta = bottom - (Canvas.GetTop(item) + item.Height);

foreach (DesignerItem di in SelectionService.GetGroupMembers(item))

{

Canvas.SetTop(di, Canvas.GetTop(di) + delta);

}

}

}

}

#endregion

#region AlignLeft Command

private void AlignLeft\_Executed(object sender, ExecutedRoutedEventArgs e)

{

var selectedItems = from item in SelectionService.CurrentSelection.OfType<DesignerItem>()

where item.ParentID == Guid.Empty

select item;

if (selectedItems.Count() > 1)

{

double left = Canvas.GetLeft(selectedItems.First());

foreach (DesignerItem item in selectedItems)

{

double delta = left - Canvas.GetLeft(item);

foreach (DesignerItem di in SelectionService.GetGroupMembers(item))

{

Canvas.SetLeft(di, Canvas.GetLeft(di) + delta);

}

}

}

}

#endregion

#region AlignHorizontalCenters Command

private void AlignHorizontalCenters\_Executed(object sender, ExecutedRoutedEventArgs e)

{

var selectedItems = from item in SelectionService.CurrentSelection.OfType<DesignerItem>()

where item.ParentID == Guid.Empty

select item;

if (selectedItems.Count() > 1)

{

double center = Canvas.GetLeft(selectedItems.First()) + selectedItems.First().Width / 2;

foreach (DesignerItem item in selectedItems)

{

double delta = center - (Canvas.GetLeft(item) + item.Width / 2);

foreach (DesignerItem di in SelectionService.GetGroupMembers(item))

{

Canvas.SetLeft(di, Canvas.GetLeft(di) + delta);

}

}

}

}

#endregion

#region AlignRight Command

private void AlignRight\_Executed(object sender, ExecutedRoutedEventArgs e)

{

var selectedItems = from item in SelectionService.CurrentSelection.OfType<DesignerItem>()

where item.ParentID == Guid.Empty

select item;

if (selectedItems.Count() > 1)

{

double right = Canvas.GetLeft(selectedItems.First()) + selectedItems.First().Width;

foreach (DesignerItem item in selectedItems)

{

double delta = right - (Canvas.GetLeft(item) + item.Width);

foreach (DesignerItem di in SelectionService.GetGroupMembers(item))

{

Canvas.SetLeft(di, Canvas.GetLeft(di) + delta);

}

}

}

}

#endregion

#region DistributeHorizontal Command

private void DistributeHorizontal\_Executed(object sender, ExecutedRoutedEventArgs e)

{

var selectedItems = from item in SelectionService.CurrentSelection.OfType<DesignerItem>()

where item.ParentID == Guid.Empty

let itemLeft = Canvas.GetLeft(item)

orderby itemLeft

select item;

if (selectedItems.Count() > 1)

{

double left = Double.MaxValue;

double right = Double.MinValue;

double sumWidth = 0;

foreach (DesignerItem item in selectedItems)

{

left = Math.Min(left, Canvas.GetLeft(item));

right = Math.Max(right, Canvas.GetLeft(item) + item.Width);

sumWidth += item.Width;

}

double distance = Math.Max(0, (right - left - sumWidth) / (selectedItems.Count() - 1));

double offset = Canvas.GetLeft(selectedItems.First());

foreach (DesignerItem item in selectedItems)

{

double delta = offset - Canvas.GetLeft(item);

foreach (DesignerItem di in SelectionService.GetGroupMembers(item))

{

Canvas.SetLeft(di, Canvas.GetLeft(di) + delta);

}

offset = offset + item.Width + distance;

}

}

}

private void Distribute\_Enabled(object sender, CanExecuteRoutedEventArgs e)

{

//var groupedItem = from item in SelectionService.CurrentSelection.OfType<DesignerItem>()

// where item.ParentID == Guid.Empty

// select item;

//e.CanExecute = groupedItem.Count() > 1;

e.CanExecute = true;

}

#endregion

#region DistributeVertical Command

private void DistributeVertical\_Executed(object sender, ExecutedRoutedEventArgs e)

{

var selectedItems = from item in SelectionService.CurrentSelection.OfType<DesignerItem>()

where item.ParentID == Guid.Empty

let itemTop = Canvas.GetTop(item)

orderby itemTop

select item;

if (selectedItems.Count() > 1)

{

double top = Double.MaxValue;

double bottom = Double.MinValue;

double sumHeight = 0;

foreach (DesignerItem item in selectedItems)

{

top = Math.Min(top, Canvas.GetTop(item));

bottom = Math.Max(bottom, Canvas.GetTop(item) + item.Height);

sumHeight += item.Height;

}

double distance = Math.Max(0, (bottom - top - sumHeight) / (selectedItems.Count() - 1));

double offset = Canvas.GetTop(selectedItems.First());

foreach (DesignerItem item in selectedItems)

{

double delta = offset - Canvas.GetTop(item);

foreach (DesignerItem di in SelectionService.GetGroupMembers(item))

{

Canvas.SetTop(di, Canvas.GetTop(di) + delta);

}

offset = offset + item.Height + distance;

}

}

}

#endregion

#region SelectAll Command

private void SelectAll\_Executed(object sender, ExecutedRoutedEventArgs e)

{

SelectionService.SelectAll();

}

#endregion

#region Helper Methods

private XElement LoadSerializedDataFromFile()

{

OpenFileDialog openFile = new OpenFileDialog();

openFile.Filter = "Designer Files (\*.xml)|\*.xml|All Files (\*.\*)|\*.\*";

if (openFile.ShowDialog() == true)

{

try

{

return XElement.Load(openFile.FileName);

}

catch (Exception e)

{

MessageBox.Show(e.StackTrace, e.Message, MessageBoxButton.OK, MessageBoxImage.Error);

}

}

return null;

}

void SaveFile(XElement xElement)

{

SaveFileDialog saveFile = new SaveFileDialog();

saveFile.Filter = "Files (\*.xml)|\*.xml|All Files (\*.\*)|\*.\*";

if (saveFile.ShowDialog() == true)

{

try

{

xElement.Save(saveFile.FileName);

}

catch (Exception ex)

{

MessageBox.Show(ex.StackTrace, ex.Message, MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

private XElement LoadSerializedDataFromClipBoard()

{

if (Clipboard.ContainsData(DataFormats.Xaml))

{

String clipboardData = Clipboard.GetData(DataFormats.Xaml) as String;

if (String.IsNullOrEmpty(clipboardData))

return null;

try

{

return XElement.Load(new StringReader(clipboardData));

}

catch (Exception e)

{

MessageBox.Show(e.StackTrace, e.Message, MessageBoxButton.OK, MessageBoxImage.Error);

}

}

return null;

}

private XElement SerializeDesignerItems(IEnumerable<DesignerItem> designerItems)

{

XElement serializedItems = new XElement("DesignerItems",

from item in designerItems

let contentXaml = XamlWriter.Save(((DesignerItem)item).Content)

select new XElement("DesignerItem",

new XElement("Left", Canvas.GetLeft(item)),

new XElement("Top", Canvas.GetTop(item)),

new XElement("Width", item.Width),

new XElement("Height", item.Height),

new XElement("ID", item.ID),

new XElement("zIndex", Canvas.GetZIndex(item)),

new XElement("IsGroup", item.IsGroup),

new XElement("ParentID", item.ParentID),

new XElement("Content", contentXaml)

)

);

return serializedItems;

}

//#####################

private List<String> GetStringListDesignerItems(IEnumerable<DesignerItem> designerItems)

{

List<string> s = (from designerItem in designerItems

select

XamlWriter.Save(((DesignerItem)designerItem).Content) + "'" +

Canvas.GetLeft(designerItem).ToString() + "'" +

Canvas.GetTop(designerItem).ToString() + "'" +

designerItem.Width.ToString() + "'" +

designerItem.Height.ToString() + "'" +

designerItem.ID.ToString() + "'" +

Canvas.GetZIndex(designerItem).ToString() + "'" +

designerItem.IsGroup.ToString()+ "'" +

designerItem.ParentID.ToString()+"'"+

designerItem.NodeType.ToString()

).ToList<string>();

return s;

}

private List<DesignerItem> FromStringDesignerItems(List<string> inputList)

{

List<DesignerItem> resdesignerItemList = new List<DesignerItem>();

foreach (String designerItemString in inputList)

{

List<string> designerItemData = designerItemString.Split('\'').ToList<string>();

Guid id = new Guid(designerItemData[5]);

DesignerItem item = new DesignerItem(id);

item.Width = Double.Parse(designerItemData[3], CultureInfo.InvariantCulture);

item.Height = Double.Parse(designerItemData[4], CultureInfo.InvariantCulture);

item.ParentID = new Guid(designerItemData[8]);

item.NodeType = (NodeTypes) Enum.Parse(typeof(NodeTypes), designerItemData[9], true);

item.IsGroup = Boolean.Parse(designerItemData[7]);

Canvas.SetLeft(item, Double.Parse(designerItemData[1]));

Canvas.SetTop(item, Double.Parse(designerItemData[2]));

Canvas.SetZIndex(item, Int32.Parse(designerItemData[6]));

Object content = XamlReader.Load(XmlReader.Create(new StringReader(designerItemData[0])));

item.Content = content;

resdesignerItemList.Add(item);

}

return resdesignerItemList;

}

private List<String> GetStringListConnections(IEnumerable<Connection> connections)

{

List<string> s = (from item in connections

select

item.Source.ParentDesignerItem.ID + " " +

item.Sink.ParentDesignerItem.ID + " " +

item.Source.Name + " " +

item.Sink.Name + " " +

item.SourceArrowSymbol + " " +

item.SinkArrowSymbol + " " +

Canvas.GetZIndex(item).ToString()

).ToList<string>();

return s;

}

private List<Connection> FromStringListToConnections(List<string> inputList)

{

List<Connection> resConnectionsList = new List<Connection>();

foreach (String connectionString in inputList)

{

List<string> connectionData = connectionString.Split(' ').ToList<string>();

Guid sourceID = new Guid(connectionData[0]);

Guid sinkID = new Guid(connectionData[1]);

String sourceConnectorName = connectionData[2];

String sinkConnectorName = connectionData[3];

Connector sourceConnector = GetConnector(sourceID, sourceConnectorName);

Connector sinkConnector = GetConnector(sinkID, sinkConnectorName);

Connection connection = new Connection(sourceConnector, sinkConnector);

Canvas.SetZIndex(connection, Int32.Parse(connectionData[6]));

resConnectionsList.Add(connection);

}

return resConnectionsList;

}

}

}

**Висновок**:

Під час виконання лабораторної роботи , я ознайомився з матричним методом представлення графічних схем алгоритмів та зробив висновок, що цей метод збереження є досить оптимальнім.