МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Вычислительная техника»

Дисциплина «Лингвистическое обеспечение САПР»

**Лабораторная работа**

**Диаграмматическая модель потоков проектной работы в виде графического языка UML-AD**

**3 вариант**

Выполнил:

студент группы ИВТАПбд-31.

Вершинин Д. В.

Проверил:

Войт Н. Н.

Ульяновск, 2018

# Цель работы:

Целью лабораторной работы является изучение лингвистического обеспечения диаграмматических моделей потоков проектных работ в виде графических языков RV-семейства.

# Порядок работы:

Для достижения указанной цели необходимо решить следующие задачи.

1.Изучить спецификацию к лингвистическому обеспечению диаграмматических моделей потоков проектных работ UML-AD.

2.Разработать собственную диаграмматическую модель потока проектных работ согласно требованиям пункта 1 в среде MSVisio для бизнес-процесса (рисунок 1).

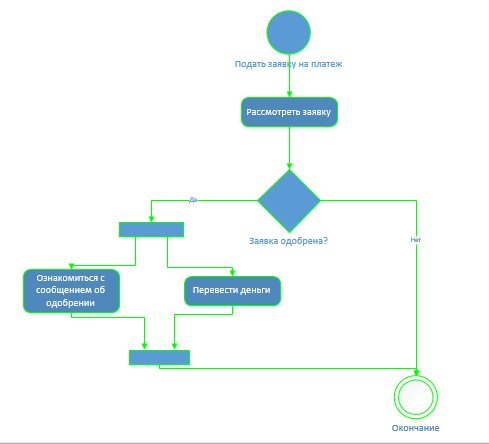


Рисунок 1 – Диаграмматическая модель бизнес-процесса

3. Разработать структуру данных в виде UMLClass, алгоритмы в виде UML-ActiveDiagram-нотаций к реализации программы.

4.Разработать исходный код на языке, согласованном с преподавателем, к реализации пункта 3.

5.Выполнить компиляцию и тестирование на наличие ошибок реализованной программы.

6.Составить детальный отчет, скриншотами программы, таблицей результатов тестирования.

# Спецификация языка UML-AD

## Элементы графической нотации UML-AD

На схеме бизнес-процесса узлы процесса можно изображать по-разному. Способ изображения узлов и переходов важен, потому что от этого зависит легкость (или сложность) понимания бизнес-процесса людьми.

Согласованные наборы графических элементов, из которых строятся схемы бизнес-процессов, называются графическими нотациями изображения бизнес-процессов.

Одной из наиболее известных графических нотаций изображения бизнес-процессов является UML Activity Diagram (далее UML AD).

В работе Stephen A. White Process Modeling Notations and Workflow Patterns, посвященной сравнению выразительной мощи UML AD и BPMN нотаций, основанной на реализациях с помощью этих нотаций типичных конструкций бизнес-процессов, содержится вывод, что выразительная мощь основных конструкций обеих нотаций примерно одинакова. Позже этот результат был подтвержден в более полном исследовании: Lauri Eloranta, Eero Kallio, Ilkka Terho A Notation Evaluation of BPMN and UML Activity Diagrams.

Диаграмма деятельности ‒ еще один способ описания поведения, который визуально напоминает старую добрую блок-схему алгоритма. Однако за счет модернизированных обозначений, согласованных с объектно-ориентированным подходом, а главное, за счет новой семантической составляющей (свободная интерпретация сетей Петри), диаграмма деятельности UML является мощным средством для описания поведения системы.

На диаграмме деятельности применяют один основной тип сущностей ‒ действие, и один тип отношений ‒ переходы (передачи управления и данных). Также используются такие конструкции как развилки, слияния, соединения, ветвления, которые похожи на сущности, но таковыми на самом деле не являются, а представляют собой графический способ изображения некоторых частных случаев многоместных отношений.

Семантически граф деятельности – это множество сущностей, которыми являются действия или деятельности, и отношения между этими сущностями, которые задают порядок их выполнения.

#### Базовые элементы нотации UML AD

##### Узел-Действие

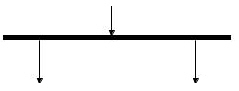


##### Маршрутные узлы

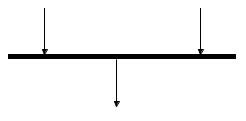
**Ветвление** - Узел выбора направления дальнейшего движения точки управления:



**Разделение** - Разделение точки управления на несколько точек управления:



**Слияние** - Слияние точек управления в одну точку управления



## 3.2. Правила и рекомендации построения диаграмм UML-AD

Диаграммы деятельности играют важную роль в понимании процессов реализации алгоритмов выполнения операций классов и потоков управления в моделируемой системе. Используемые для этой цели традиционные блок-схемы алгоритмов обладают серьёзными ограничениями в представлении параллельных процессов и их синхронизации. Применение дорожек и объектов открывает дополнительные возможности для наглядного представления бизнес-процессов, позволяя специфицировать деятельность подразделений компаний и фирм.

Содержание диаграммы деятельности во многом напоминает диаграмму состояний, хотя и не тождественно ей. Поэтому многие рекомендации по построению последней оказываются справедливыми применительно к диаграмме деятельности. В частности, эта диаграмма строится для отдельного класса, варианта использования, отдельной операции класса или целой подсистемы.

С одной стороны, на начальных этапах проектирования, когда детали реализации деятельностей в проектируемой системе неизвестны, построение диаграммы деятельности начинают с выделения под-деятельностей, которые в совокупности образуют деятельность подсистем. В последующем, по мере разработки диаграмм классов и состояний, эти под-деятельности уточняются в виде отдельных вложенных диаграмм деятельности компонентов подсистем, какими выступают классы и объекты.

С другой стороны, отдельные участки рабочего процесса в существующей системе могут быть хорошо отлаженными, и у разработчиков может возникнуть желание сохранить этот механизм выполнения действий в проектируемой системе. Тогда строится диаграмма деятельности для этих участков, отражающая конкретные особенности выполнения действий с использованием дорожек и объектов. В последующем такая диаграмма вкладывается в более общие диаграммы деятельности для подсистемы и системы в целом, сохраняя свой уровень детализации.

Таким образом, процесс объектно-ориентированного анализа и проектирования сложных систем представляется как последовательность итераций нисходящей и восходящей разработки отдельных диаграмм, включая и диаграмму деятельности. Доминирование того или иного из направлений разработки определяется особенностями конкретного проекта и его новизной.

В случае типового проекта большинство деталей реализации действий могут быть известны заранее на основе анализа существующих систем или предшествующего опыта разработки систем-прототипов. Для этой ситуации доминирующим будет восходящий процесс разработки (Зачем изобретать велосипед заново?). Использование типовых решений может существенно сократить время разработки и избежать возможных ошибок при реализации проекта.

При разработке проекта новой системы, процесс функционирования которой основан на новых технологических решениях, ситуация представляется более сложной. А именно, до начала работы над проектом могут быть неизвестны не только детали реализации отдельных деятельностей, но и само содержание этих деятельностей становится предметом разработки. В данном случае доминирующим будет нисходящий процесс разработки от более общих схем к уточняющим их диаграммам. При этом достижение такого уровня детализации всех диаграмм, который достаточен для понимания особенностей реализации всех действий и деятельностей, может служить признаком завершения отдельных этапов работы над проектом.

В заключение следует заметить, что диаграмма деятельности, так же, как и другие виды канонических диаграмм, не содержит средств выбора оптимальных решений. При разработке сложных проектов проблема выбора оптимальных решений становится весьма актуальной. Рациональное расходование средств, затраченных на разработку и эксплуатацию системы, повышение ее производительности и надежности зачастую определяют конечный результат всего проекта. В такой ситуации можно рекомендовать использование дополнительных средств и методов, ориентированных на аналитико-имитационное исследование моделей системы на этапе разработки ее проекта.

В частности, при построении диаграмм деятельности сложных систем могут быть успешно использованы различные классы сетей Петри (классические, логико-алгебраические, стохастические, нечеткие и др.) и нейронных сетей. Применение этих формализмов позволяет не только получить оптимальную структуру поведения системы на ее модели, но и специфицировать целый ряд дополнительных характеристик системы, которые не могут быть представлены на диаграмме деятельности и других диаграммах UML.

# 4. Построение диаграммы

Для построения диаграммы UML-AD на основе диаграммы (рисунок 1) и дальнейшей работы с ней, требуется выбрать базовые элементы фигур с добавлением свойств у каждой фигуры.

Конечный набор фигур выглядит следующим образом:

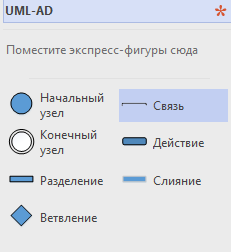


Рисунок 2 – Набор элементов для UML-AD

На основе созданного набора элементов строим UML-AD диаграмму.

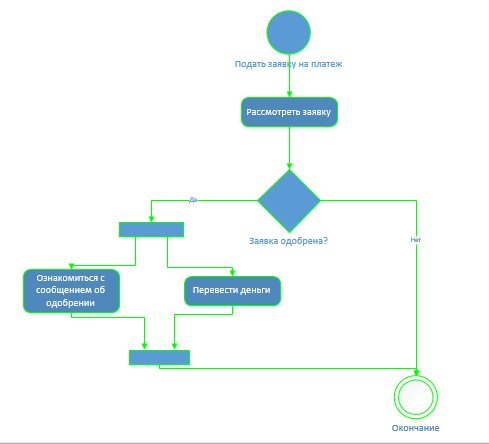


Рисунок 3 – Построенная UML-AD диаграмма

# 5. Диаграмма классов

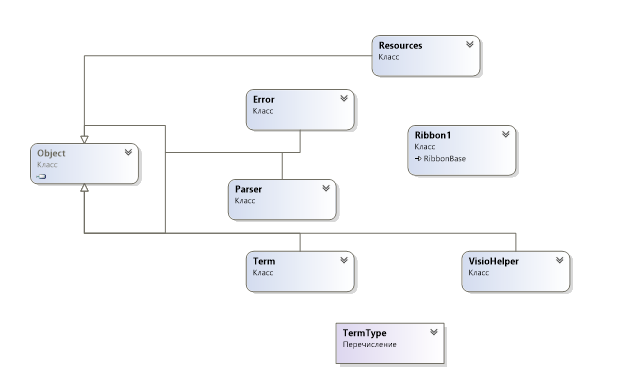


Рисунок 4 – Структура данных в виде UML Class

# 6. Компиляция и тестирование на наличие ошибок реализованной программы

|  |  |
| --- | --- |
| Наименование ошибки | Реакция программы |
| Отсутствие начального события |  |
| Отсутствие завершающего события |  |
| Несоединенный элемент |  |
| Deadlock |  |
| Неверное количество соединений |  |

# 7. Описание кода

Структура приложения включает в себя следующие классы:

1) Parser.cs – в классе производиться считывание фигур и их параметров. Также, для каждой фигуры осуществляется запись следующей и предшествующей фигур.

2) Term.cs – определяются переменные и массивы для работы с фигурами.

Метод Name – метод, который возвращает имя фигуры.

Метод Type – метод, который возвращает тип фигуры.

Метод Term – метод, который записывает характеристики каждой фигуры в Term.

Метод SetType – метод, который задает к каждой фигуре нужный тип термы.

Метод GetNextIdIndex – метод, который возвращает индекс следующей фигуры.

3) TermType.cs – класс- перечисление, содержащий типы фигур.

4) Ribon1.cs – основной класс, при нажатии кнопки «Анализировать» в Visio выполняет проверку схемы UML-AD на ошибки.

5) Error.cs – класс, который определяет поля *ShapeId и Name* у List<Error> ошибок Error.

6) VisioHelper.cs – класс, с помощью которого производится работа с API Visio.

Метод ReadANamedCustomProperty() – метод, в котором производится чтение свойств у нарисованных фигур.

Метод changeColor() – метод, в котором производиться замена цвета блоков в Visio.

# Вывод

В ходе данной лабораторной работы был изучен язык UML-AD, получены навыки построения UML-AD-диаграмм в MS Visio, а также создана программа на языке С#, проверяющая созданную в MS Visio диаграмму на ошибки.

# Исходный код:

Класс Ribon1.cs:

...

private void Analyze\_Click(object sender, RibbonControlEventArgs e) {

bool flagInclusiveGateway = false;

int count = 0;

bool flagEnd = true;

var shapes = VisioAddInTest.Globals.ThisAddIn.Application.ActivePage.Shapes;

var terms = new List<Term>();

var errors = new List<Error>();

Parser.Run(shapes, terms);

Dictionary<int, int> CurrentVisitCount = new Dictionary<int, int>();

Dictionary<int, int> TotalVisitCount = new Dictionary<int, int>();

List<Term> CommonStack = new List<Term>();

foreach (var term in terms) {

CurrentVisitCount[term.Id] = 0;

TotalVisitCount[term.Id] = 0;

}

var startTerms = terms.Where(t => t.Type.ToString().Equals(TermType.Start.ToString())).ToList();

if (startTerms.Count == 0) {

errors.Add(new Error(0, "Нет начального элемента"));

}

Else {

var currentTerm = startTerms[0];

var state = 0;

while (state != 7) {

currentTerm.isChecked = true;

switch (state) {

case 0:

if (currentTerm.Type.ToString().Equals(TermType.Start.ToString())) {

currentTerm.isChecked = true;

CommonStack.AddRange(startTerms);

state = 2;

}

break;

case 1:

if (currentTerm.Type.ToString().Equals(TermType.Rel.ToString())) {

int nextIndex = currentTerm.GetNextTermId();

try {

currentTerm = terms.Where(t => t.Id == nextIndex).First();

}

catch {

errors.Add(new Error(currentTerm.Id, "Несоединенный элемент"));

state = 6;

break;

}

state = 4;

}

break;

case 2:

if (currentTerm.Type.ToString().Equals(TermType.Start.ToString())) {

if (count > 1 && flagInclusiveGateway) {

CommonStack.Reverse();

errors.Add(new Error(count, "Deadlock"));

Console.WriteLine("Deadlock");

state = 6;

break;

}

currentTerm = CommonStack.Last();

if (currentTerm.GetNextIdIndex() >= currentTerm.NextTermsId.Count() - 1) {

CommonStack.Reverse();

CommonStack.Remove(currentTerm);

CommonStack.Reverse();

}

int nextIndex = currentTerm.GetNextTermId();

if (nextIndex == -1) {

state = 5;

break;

}

try {

currentTerm = terms.Where(t => t.Id == nextIndex).First();

}

catch {

errors.Add(new Error(currentTerm.Id, "Несоединенный элемент"));

state = 6;

break;

}

state = 1;

}

if (currentTerm.Type.ToString().Equals(TermType.Action.ToString()) || currentTerm.Type.ToString().Equals(TermType.InterEventTimer.ToString())) {

if (currentTerm.Type.ToString().Equals(TermType.Action.ToString()) && flagInclusiveGateway && count > 0) {

flagInclusiveGateway = false;

count = 0;

}

currentTerm = CommonStack.Last();

if (currentTerm.GetNextIdIndex() >= currentTerm.NextTermsId.Count() - 1) {

CommonStack.Reverse();

CommonStack.Remove(currentTerm);

CommonStack.Reverse();

}

int nextIndex = currentTerm.GetNextTermId();

if (nextIndex == -1) {

state = 5;

break;

}

try {

currentTerm = terms.Where(t => t.Id == nextIndex).First();

}

catch {

errors.Add(new Error(currentTerm.Id, "Несоединенный элемент"));

state = 6;

break;

}

state = 1;

}

if (currentTerm.Type.ToString().Equals(TermType.ExclusiveGateway.ToString())) {

if (count > 1 && flagInclusiveGateway) {

CommonStack.Reverse();

errors.Add(new Error(count, "Deadlock"));

Console.WriteLine("Deadlock");

state = 6;

break;

}

currentTerm = CommonStack.Last();

if (currentTerm.GetNextIdIndex() >= currentTerm.NextTermsId.Count() - 1) {

CommonStack.Reverse();

CommonStack.Remove(currentTerm);

CommonStack.Reverse();

}

int nextIndex = currentTerm.GetNextTermId();

if (nextIndex == -1) {

state = 5;

break;

}

try {

currentTerm = terms.Where(t => t.Id == nextIndex).First();

}

catch {

errors.Add(new Error(currentTerm.Id, "Несоединенный элемент"));

state = 6;

break;

}

state = 1;

}

if (currentTerm.Type.ToString().Equals(TermType.ParallelGateway.ToString()) || currentTerm.Type.ToString().Equals(TermType.InclusiveGateway.ToString())) {

if (currentTerm.Type.ToString().Equals(TermType.InclusiveGateway.ToString()) && flagInclusiveGateway && count > 0) {

flagInclusiveGateway = false;

count = 0;

}

else if (currentTerm.Type.ToString().Equals(TermType.ParallelGateway.ToString()) && count > 1 && flagInclusiveGateway) {

CommonStack.Reverse();

errors.Add(new Error(count, "Deadlock"));

Console.WriteLine("Deadlock");

state = 6;

break;

}

else if (currentTerm.Type.ToString().Equals(TermType.InclusiveGateway.ToString())) {

flagInclusiveGateway = true;

count = currentTerm.Id;

}

var flag = false;

CommonStack.Reverse();

foreach (var sTerm in CommonStack) {

if ((sTerm.Type.ToString().Equals(TermType.ParallelGateway.ToString()) || sTerm.Type.ToString().Equals(TermType.InclusiveGateway.ToString())) && CurrentVisitCount[sTerm.Id] != TotalVisitCount[sTerm.Id]) {

continue;

}

else {

if (sTerm.GetNextIdIndex() >= sTerm.NextTermsId.Count() - 1) {

CommonStack.Remove(sTerm);

}

int nextIndex = sTerm.GetNextTermId();

if (nextIndex == -1) {

state = 5;

flag = true;

break;

}

if (flag) {

break;

}

try {

currentTerm = terms.Where(t => t.Id == nextIndex).First();

}

catch {

errors.Add(new Error(sTerm.Id, "Несоединенный элемент"));

state = 6;

break;

}

state = 1;

flag = true;

break;

}

}

CommonStack.Reverse();

if (flag) {

break;

}

CommonStack.Reverse();

errors.Add(new Error(currentTerm.Id, "Deadlock"));

Console.WriteLine("Deadlock");

state = 6;

break;

}

break;

case 3:

break;

case 4:

if (currentTerm.Type.ToString().Equals(TermType.Action.ToString()) || currentTerm.Type.ToString().Equals(TermType.InterEventTimer.ToString())) {

CommonStack.Add(currentTerm);

if (CurrentVisitCount[currentTerm.Id] != 0 && CurrentVisitCount[currentTerm.Id] != TotalVisitCount[currentTerm.Id]) {

CurrentVisitCount[currentTerm.Id]++;

}

if (CurrentVisitCount[currentTerm.Id] == 0) {

CurrentVisitCount[currentTerm.Id] = 1;

TotalVisitCount[currentTerm.Id] = currentTerm.PrevTermsId.Count();

}

state = 2;

}

if (currentTerm.Type.ToString().Equals(TermType.ExclusiveGateway.ToString())) {

CommonStack.Add(currentTerm);

if (CurrentVisitCount[currentTerm.Id] != 0 && CurrentVisitCount[currentTerm.Id] != TotalVisitCount[currentTerm.Id]) {

CurrentVisitCount[currentTerm.Id]++;

}

if (CurrentVisitCount[currentTerm.Id] == 0) {

CurrentVisitCount[currentTerm.Id] = 1;

TotalVisitCount[currentTerm.Id] = currentTerm.PrevTermsId.Count();

}

state = 2;

}

if (currentTerm.Type.ToString().Equals(TermType.ParallelGateway.ToString()) || currentTerm.Type.ToString().Equals(TermType.InclusiveGateway.ToString())) {

if (!CommonStack.Contains(currentTerm)){

CommonStack.Add(currentTerm);

}

if (CurrentVisitCount[currentTerm.Id] != 0 && CurrentVisitCount[currentTerm.Id] != TotalVisitCount[currentTerm.Id]) {

CurrentVisitCount[currentTerm.Id]++;

}

if (CurrentVisitCount[currentTerm.Id] == 0) {

CurrentVisitCount[currentTerm.Id] = 1;

TotalVisitCount[currentTerm.Id] = currentTerm.PrevTermsId.Count();

}

state = 2;

}

if (currentTerm.Type.ToString().Equals(TermType.End.ToString())) {

if (CurrentVisitCount[currentTerm.Id] != 0 && CurrentVisitCount[currentTerm.Id] != TotalVisitCount[currentTerm.Id]) {

CurrentVisitCount[currentTerm.Id]++;

}

if (CurrentVisitCount[currentTerm.Id] == 0) {

CurrentVisitCount[currentTerm.Id] = 1;

TotalVisitCount[currentTerm.Id] = currentTerm.PrevTermsId.Count();

}

state = 5;

flagEnd = false;

}

break;

case 5:

if (CommonStack.Count() > 0) {

currentTerm = CommonStack.Last();

state = 2;

break;

}

state = 6;

break;

case 6:

var MissTerms = CurrentVisitCount.Where(t => TotalVisitCount[t.Key] != t.Value).ToDictionary(t => t.Key, t => t.Value);

if (MissTerms.Count() != 0) {

errors.AddRange(MissTerms.Select(mt => new Error(mt.Key, "Неверное количество соединений")));

}

state = 7;

break;

}

}

}

errors.AddRange(terms.Where(t => t.isChecked == false).Select(t => new Error(t.Id, "Несоединенный элемент")).ToList());

if (flagEnd)

errors.Add(new Error(1, "Нет завершающего элемента"));

foreach (Microsoft.Office.Interop.Visio.Shape shape in shapes) {

VisioHelper.changeColor(shape, "RGB(0,255,0)");

}

if (errors.Count() > 0) {

var \_form1 = new Form1(VisioAddInTest.Globals.ThisAddIn.Application.ActivePage, errors);

\_form1.Show();

}

}

}

}Класс Term.cs:

…

private TermType SetType(string value) {

if (value.Equals("Начальное событие"))

return TermType.Start;

else if (value.Equals("Завершающее событие"))

return TermType.End;

else if (value.Equals("Таймер"))

return TermType.InterEventTimer;

else if (value.Equals("Подпроцесс"))

return TermType.Action;

else if (value.Equals("Поток управления"))

return TermType.Rel;

else if (value.Equals("Оператор исключающего ИЛИ"))

return TermType.ExclusiveGateway;

else if (value.Equals("Оператор И"))

return TermType.ParallelGateway;

else if (value.Equals(" Оператор включающего ИЛИ"))

return TermType.InclusiveGateway;

return TermType.None;

}

…

Класс TermType.cs:

…

public enum TermType

{

None,

Start,

Action,

InterEventTimer,

ExclusiveGateway,

ParallelGateway,

Rel,

End,

InclusiveGateway

}

…