

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра Системного Программирования

Саравайский Михаил Александрович

Компонента для визуализации графов с
сохранением связи с исходным
представлением данных

Курсовая работа

Научный руководитель:
аспирант кафедры Системного программирования Григорьев С. В.

Санкт-Петербург
2013

Оглавление

Введение	3
1. Постановка задачи	4
2. Обзор существующих решений	5
3. Особенности реализации	8
4. Апробация библиотеки на простой среде разработки	12
Заключение	16

Введение

При разработке программного обеспечения постоянно возникают большие объемы информации, с которой приходится работать человеку. Зачастую, такая информация представляется текстом. Люди по своей природе же работают с текстом куда хуже чем с информацией, представленной графически. К счастью, почти всегда данные в разработке программного обеспечения можно представить в виде графа, так как в основном приходится иметь дело с такими структурами данных, как деревья разбора и графы потока вызовов. Поэтому удобным инструментом для работы с этими данными была бы библиотека, которая позволяла бы по изначальным данным строить и визуализировать граф, элементы которого сохраняют связь с исходными структурами. Причем, последнее является чрезвычайно важным элементом, так как найдя необходимый элемент в графе и не имея этой обратной связи, приходилось бы снова искать этот элемент в исходном представлении и терялась бы основная идея данной работы. На данный момент существует несколько решений, которые будут рассмотрены ниже, которые предоставляют некоторый требуемый функционал, однако ни одно из них не является предназначенным именно для такого использования, то есть либо возможности сохранения связей визуализации с исходным представлением существуют, но эти связи существуют только на уровне представление графа - визуализации, либо же таких механизмов нет. Целью данной работы является создание инструмента, первоочередной задачей которого будет визуализация данных, с сохранением связи с изначальным представлением этих данных. Такой инструмент сделает анализ таких структур, как абстрактные синтаксические деревья или графы потока управления удобным и понятным. Например, возможно будет рассмотреть дерево разбора какой-либо программы, посмотреть на его структуру, а затем переместиться в код проблемного места прямо из графа.

1. Постановка задачи

От конечного решения требуется:

- Визуализировать граф исходя из данных, предоставленных пользователем
- Предоставлять информацию о действиях с графом в смысле изначального представления
- Обеспечить простоту как использования разработанной библиотеки для разработчиков, так и готовых продуктов для конечных пользователей

Так же хорошим бонусом являлась бы поддержка WPF, что сделало бы библиотеку привлекательной для разработчиков, которые строят визуально привлекательные приложения. А вместе с тем и поддержка Silverlight, что позволило бы создавать на основе готовой библиотеки веб-приложения с широкими возможностями визуализации графов.

В качестве платформы предполагается выбрать .NET Framework. Данный выбор обуславливается перспективностью данной платформы достаточно быстрым развитием. В связи с данным выбором, далее, предполагается рассматривать исключительно рынок приложений и инструментов в рамках платформы .NET.

2. Обзор существующих решений

Перед началом разработки требуется изучить существующие на данный момент аналогичные решения, такие как:

1. Microsoft Automatic Graph Layout [3]
2. yFiles for .NET [4]
3. GraphSharp [1]

После этого либо доработать какое-либо из существующих решений, либо разработать собственное, максимально отвечающее требованиям.

Было рассмотрено несколько существующих на данный момент решений, которые позволяют работать с графами.

MSAGL

Известным решением для отрисовки графов является Microsoft Automatic Graph Layout, однако данный открытый проект был заброшен разработчиками (Microsoft) и доступен только как платный продукт по достаточно большой цене. Предполагается, что собственная доработка невозможна. Из описания и готовых примеров следует, что данный продукт хоть и подразумевает наличие обратных связей с внутренним представлением, однако данные связи ограничены встроенным представлением конкретного продукта, чего недостаточно, так как предполагается поддержка связи с любыми исходными данными. Но, сам факт существования данного продукта говорит, что данное направление в разработке достаточно перспективно, поскольку такая компания как Microsoft и такие работники, как работники Microsoft Research занимались производством компоненты по визуализации графов.

yFiles

Был произведен обзор таких решений как yFiles for .NET. Говоря о yFiles, сразу стоит сказать, что предоставляемые возможности весьма широки. С точки зрения автоматизированного анализа графов, данный инструмент предоставляет широчайшие возможности и имеет встроенные механизмы реализации различных алгоритмов на графах.

С точки зрения визуализации данный инструмент позволяет:

- Создавать графы
- Редактировать графы на лету
- Удобно навигироваться в графе
- Поддерживает умные “магнитные” алгоритмы просмотра вершин. Под магнитными подразумевается возможность вести указатель, который будет Автоматически идти по ребру выходящему из конкретного узла
- Применять различные алгоритмы раскладки графа
- Визуализировать древовидные структуры с поддержкой свертки вышестоящих узлов

У данного проекта существует WPF(Windows Presentation Foundation) сборка, что позволяет встраивать элементы с визуализацией в WPF приложения. Вместе с этим существует и SilverLight сборка, что дает возможность встраивать элементы визуализации графов в веб-приложения.

Однако, распространяется данный продукт по достаточно большой цене, а исходные коды распространяются по еще большей цене.

graphSharp

Последним среди обзореваемых была рассмотрена такая библиотека как graphSharp. graphSharp основан на представлении данных с использованием QuickGraph, однако предоставляет механизмы для визуализации. Данное решение больше всего подходит для поставленных целей по модели распространения, так как проект является открытым и его исходный код размещен в репозитории на codeplex.com. Однако, вместе с явным преимуществом существует так же очень сильный недостаток в виде полного отсутствия какой-либо документации. Однако возможности данной библиотеки в визуализации графов широки. Поддерживаются:

- Различные алгоритмы раскладки графов(в связи с открытостью возможно добавление своих)
- Различные пользовательские настройки вида отображения

Существует не только WPF сборка, но и SilverLight сборка, что позволяет встраивать элементы визуализирования не только в Windows-приложения, но и в Веб-приложения.

Приведем таблицу сравнительного анализа по ключевым пунктам (Таблица 1)

Таблица 1: таблица сравнения существующих решений

Продукт	привычная навигация	Различные раскладки	WPF + Silverlight	Стоимость
MSAGL	+	+	-	280
yFiles	+	+	+	> 50000
GraphSharp	-	+	+	свободно

В ходе анализа было принято решение расширять возможности graphSharp, так как данная библиотека больше всего подходит по большинству параметров. Она является открытой, что удобно для доработки необходимой функциональности. Предполагается размещать исходный код в том же репозитории, где существует проект, создав отдельную ветку разработки.

3. Особенности реализации

В данном разделе будут рассмотрены подходы, примененные в разработке и решения принятые в ходе работ.

Прежде всего стоит сказать, что в современных условиях, когда мощности компьютера пятилетней давности умещаются в смартфоне и всегда могут находиться под рукой, а мощности рабочих станций продолжают расти в соответствии с законом Мура, эффективность и оптимальность простой визуализации отступают на второй план перед визуальной ясностью и простотой работы.

Достаточно много времени было потрачено на понимание основных механизмов, которые лежат в основе graphSharp, так как отсутствует какая-либо документация. Дальнейшим шагом разработки стало проектирование взаимодействий пользователей с инструментом. Были предприняты попытки создать универсальный алгоритм генерации графов (в виде структур QuickGraph) из любых исходных данных, которые можно представить в виде графов, однако сразу стало понятно, что данные могут иметь совершенно разное представление и структуру, которые невозможно как-либо предугадать. Поэтому от идеального для пользователей подхода (рис.1) пришлось отказаться и перейти к рассмотрению менее привлекательной, но более реальной идеи (рис. 2), переложить на пользователя работу по переводу данных в структуру графа с сохранением информации об изначальном представлении.



Рис. 1: Автоматическая генерация графа

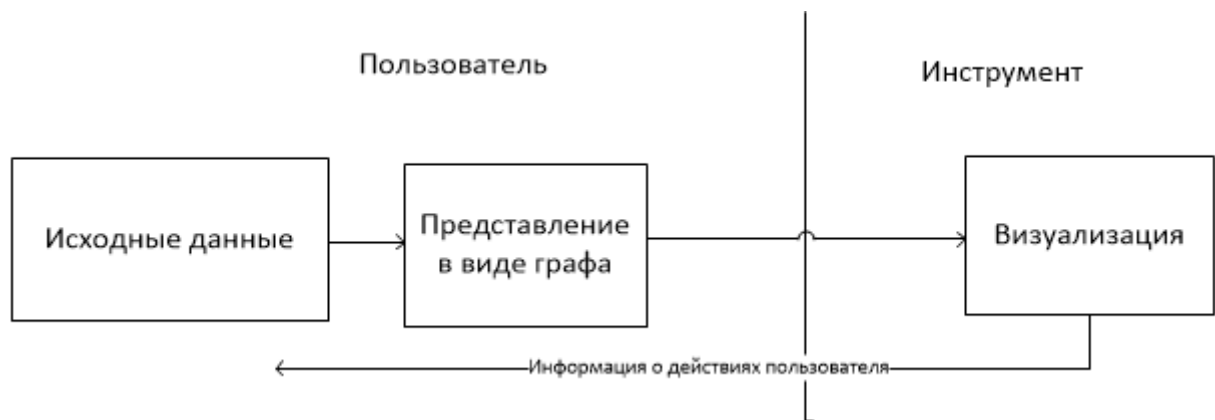


Рис. 2: Ручная генерация графа

Следующим вопросом стал формат принимаемых данных. Так как в основе используемой библиотеки лежит представление QuickGraph, очевидно, хорошим подходом является прием данных, представленных в виде графа в формате QuickGraph, однако, даже такое уточнение оставляет вопрос о конкретном виде открытым. Самые общие описание графов QuickGraph лишены какой-либо информации о сути данных. Поэтому было решено создать более узкий класс, который является наследником общих структур QuickGraph и оберткой для произвольных данных. Такой подход позволяет и использовать все возможности graphSharp для данного представления, и сохранять информацию об изначальном представлении.

Конечным результатом работы ожидается разработка методов взаимодействия с инструментом. В связи с тем, что принципы разработки в соответствии с паттерном MVC широко применяются в различных WPF приложениях и в самой graphSharp, в ходе небольших размышлений было принято решение о введении объекта контроллера, который управлял бы необходимыми графами и элементами визуализации. Такой подход позволит конечному пользователю не утруждать себя отслеживанием нескольких объектов и взаимодействовать исключительно с контроллером, который будет визуализировать графы и предоставлять элементы и информацию о действиях в них.

Таким образом модель взаимодействия предполагается следующая (рис. 3):

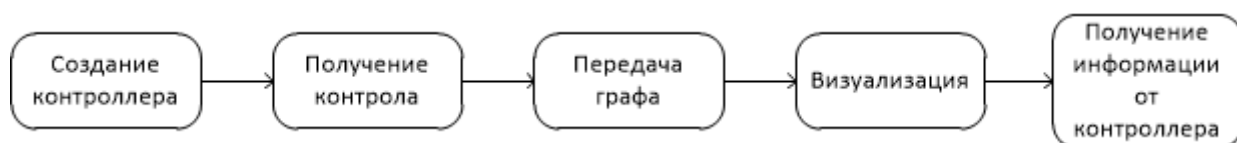


Рис. 3: Схема взаимодействия

Данный подход сохранит ясность для дальнейшей разработки, а так же облегчит взаимодействие с инструментом для пользователей. Помимо этого, так же это позволит пользователю с своих продуктах одновременно легко поддерживать визуализацию различных данных в различных элементах.

Рассмотрим простой пример. Существует некоторый исходный код и из него можно получить как минимум две древовидные структуры: дерево разбора и граф потока управления. Допустим, что пользователь хочет визуализировать оба графа. Для этого будет достаточно следующих шагов:

- Создать контроллер
- Вызвать методы контроллера для создания элементов визуализации
- Разместить эти элементы в своем приложении
- Сконструировать графы в терминах структур-обертки QuickGraph
- Передать эти графы в контроллер
- Подписаться на события контроллера

Причем порядок действий можно в зависимости от ситуации изменять, например, ускорять работу по визуализации, заранее передавая граф контроллеру, а затем уже готовый результат помещать в приложение.

Далее спустимся к деталям реализации, в общем взаимодействие компоненты инструмента можно увидеть на следующей диаграмме классов (рис.4)

Таким образом единственным неразрешенным вопросом остается метод получения информации о действиях пользователя с графом для дальнейшей передачи этой информации пользователю. Однако, так как для визуализации graphSharp применяет лучшие практики разработки WPF приложений и каждая конкретная вершина при визуализации заключается в отдельный элемент то достаточно модифицировать шаблон визуализации конкретного типа и перехватывать события нажатий кнопок мыши. Такие события содержат помимо информации о нажатых кнопках так же информацию об объекте, который вызвал данное событие, а так как шаблон визуализации содержит связь с исходной вершиной, которую и можно выдать пользователю.

Основные работы по созданию данной библиотеки можно свести к модификации graphSharp и QuickGraph таким образом, чтобы структуры, отвечающие за представление графа в памяти оборачивали исходные данные, а визуальная раскладка графа сообщала наружу об определенных действиях.

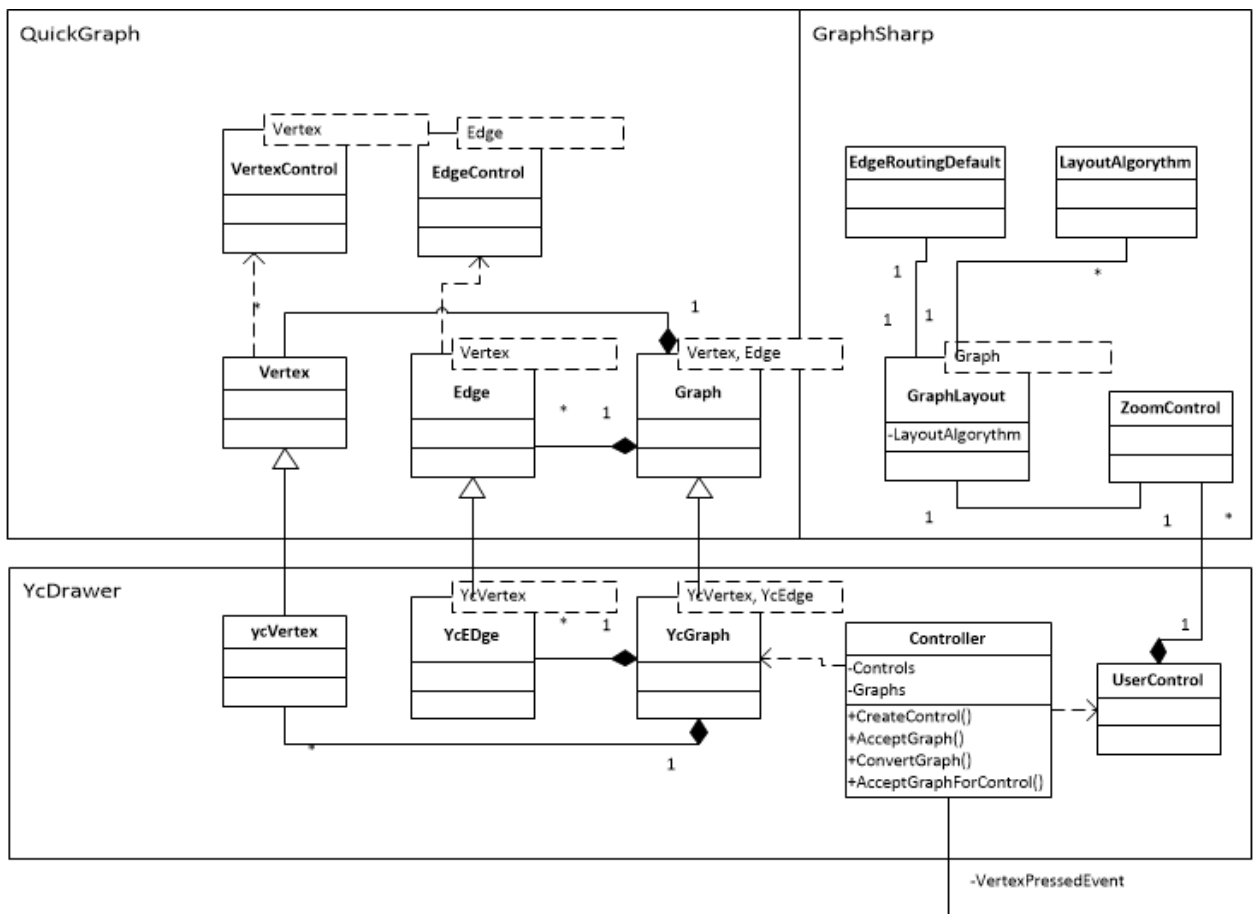


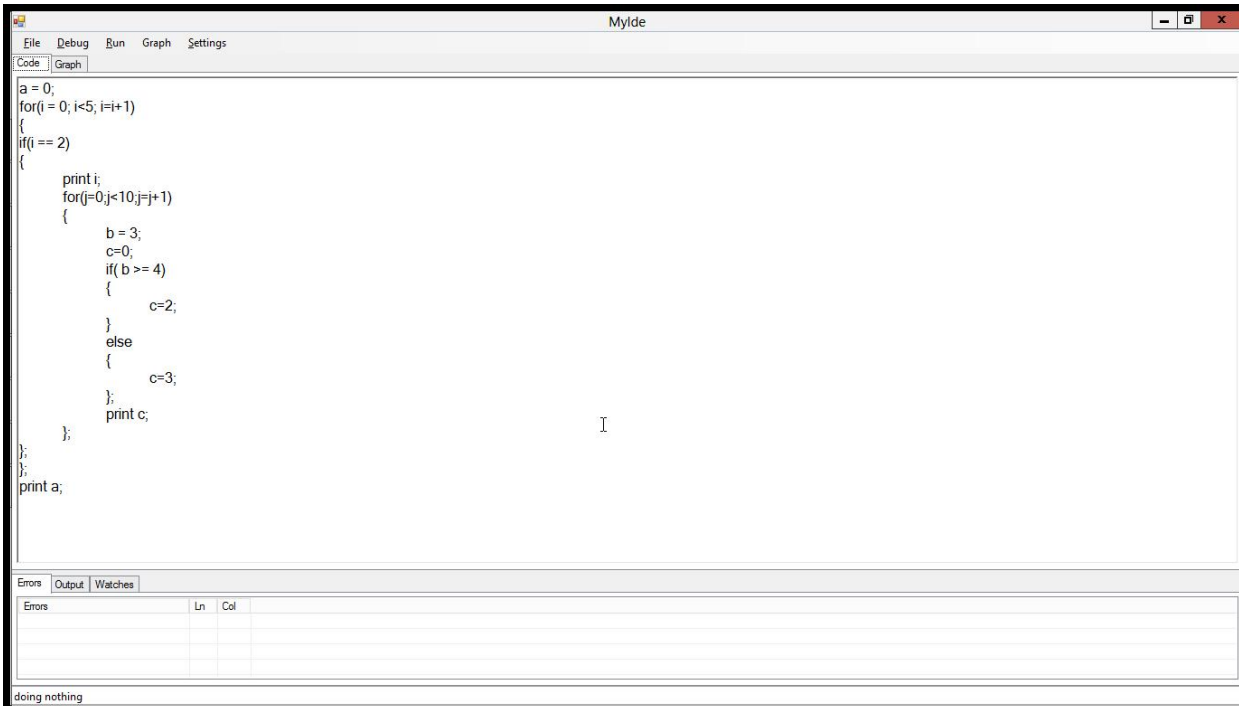
Рис. 4: Диаграмма классов библиотеки

В конечном итоге разработанный инструмент предоставляет возможность используя только один объект контроллер визуализировать различные данные, используя для этого различные элементы отображения и отслеживать действия, которые совершает пользователь с визуальным представлением. Причем за обработку различных эффектов отвечают методы разработанные в рамках проекта graphSharp, а за эффективное представление графов в памяти отвечают структуры QuickGraph.

4. Апробация библиотеки на простой среде разработки

Для примера использования было создано демо-приложение, простая среда разработки для просто C-подобного языка. В данную среду разработки был встроен элемент визуализации разработанной библиотеки и добавлены методы генерации представления в виде графа дерева разбора и обработки действий пользователя.

Схема работы предполагается следующая: пользователь пишет какой-либо код (рис. 5)



```
File  Debug  Run  Graph  Settings
Code  Graph
a = 0;
for(i = 0; i < 5; i = i + 1)
{
  if(i == 2)
  {
    print i;
    for(j = 0; j < 10; j = j + 1)
    {
      b = 3;
      c = 0;
      if(b >= 4)
      {
        c = 2;
      }
      else
      {
        c = 3;
      }
      print c;
    }
  }
};
print a;
```

Errors Output Watches

Errors	Ln	Col

doing nothing

Рис. 5: Некоторый код в среде разработки

Такой код должен быть валидным. Валидность кода можно проверить запустив синтаксический разбор. Затем пользователь нажимает кнопку построить граф и автоматически переключается на вкладку с визуализированным графом (рис. 6). На этом шаге можно выбрать более подходящий алгоритм раскладки графа (например, в нашем случае - Tree) и тут же увидеть новый результат (рис. 7) Так же возможно приблизить или переместить всю раскладку по желанию (рис. 8) После этого пользователь может нажать на необходимую вершину и в коде будет подсвечен соответствующий элемент (рис. 9)



Рис. 6: Граф дерева разбора

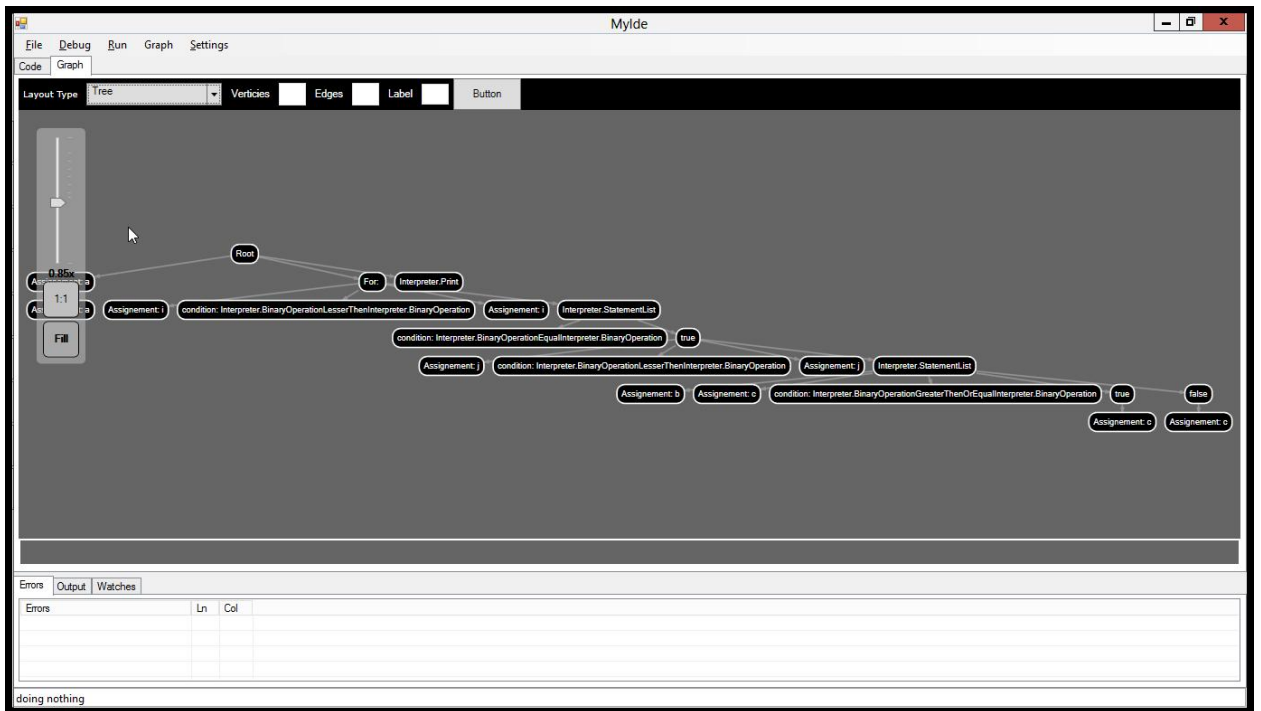


Рис. 7: Дерево разбора в правильной раскладке

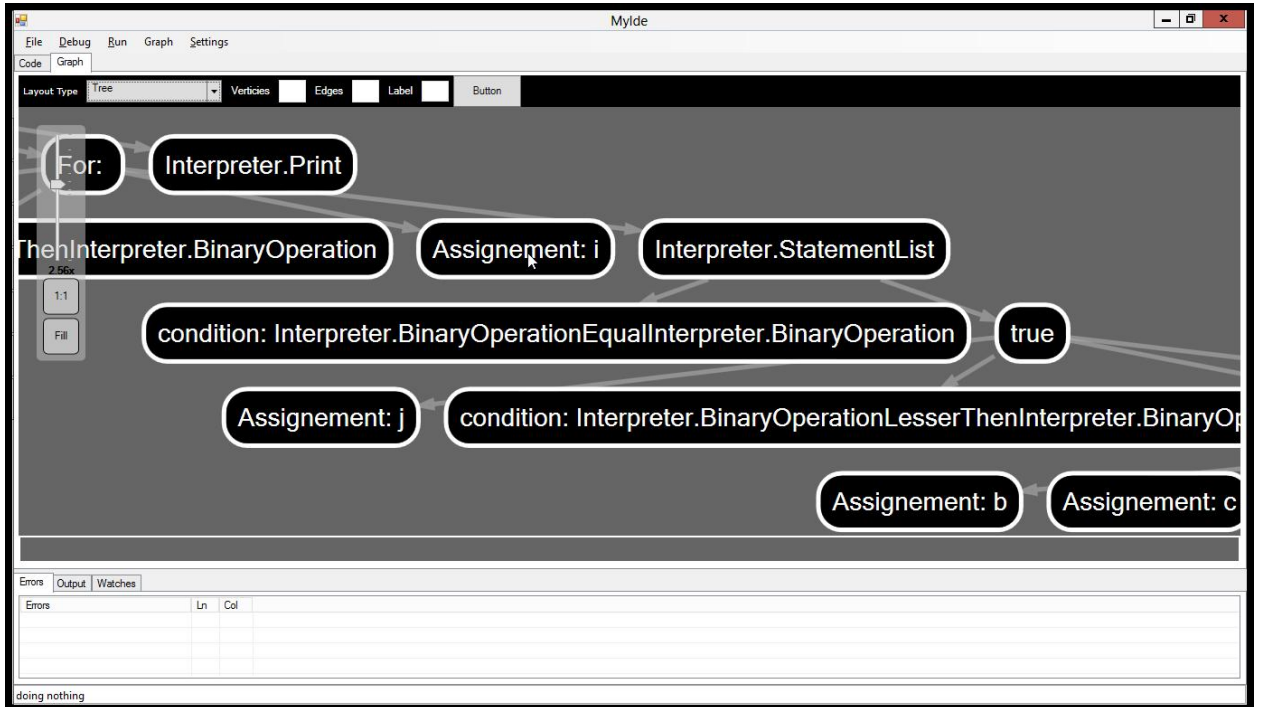


Рис. 8: Приближение к необходимой вершине

```

a = 0;
for(i = 0, i < 5; i++)
{
  if(i == 2)
  {
    print i;
    for(j=0; j < 10; j++)
    {
      b = 3;
      c = 0;
      if(b >= 4)
      {
        c = 2;
      }
      else
      {
        c = 3;
      };
      print c;
    };
  };
};
print a;

```

Рис. 9: Навигация обратно в код

Самым трудоемким моментом в переработке существовавшей ранее среды разработки стал процесс доопределение текстовой информации в вершинах дерева разбора для удобства чтения, тогда как компоновка графа и отслеживание событий в графе заняли буквально меньше сотни строк кода.

Заключение

В результате работы была доработана библиотека GraphSharp, для перехвата действий пользователя. Были созданы структуры-обертки для пользовательских данных, что позволяет сохранять связь с исходным представлением. Библиотека, получившаяся в результате работы позволяет следующее:

- Визуализация данных с сохранением связи с исходным представлением
- Выдача информации о действиях пользователя
- Управление различными элементами через единый объект-контроллер

Данный инструмент предоставляет WPF элементы, что позволяет встраивать компоненты визуализации в WPF приложения, что делает конечные приложения, написанные с использованием данного инструмента более привлекательными для пользователя. Библиотеку вместе с примером можно найти в репозитории на codeplex.com по адресу <http://graphsharp.codeplex.com/SourceControl/network/forks/gsv/graphsharp>

Список литературы

- [1] CodePlex, GraphSharp project, <http://goo.gl/H64oh>, 2007.
- [2] CodePlex, QuickGraph project, <http://goo.gl/s2J9e>, 2013.
- [3] Microsoft Research, MSAGL Documentation, <http://goo.gl/JbEJs>, 2012.
- [4] yWorks, yFiles, <http://goo.gl/7VAhJ>, 2013.