**4 КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

В разделе 3 описана разработка архитектуры программного средства, включающая общий вид его структуры и порядок функционирования. Таким образом, определена основа, достаточная для реализации программы в виде исполняемого кода. Обычным средством для реализации является платформа разработки, позволяющая программистам работать с удобным набором команд вместо бинарного машинного кода.

**4.1** Выбор средств разработки

В подразделе 3.1 был определен общий вид программы - оконное приложение, а также характер выполняемых работ – обработка графической информации. Из этого следует ряд требований, определяющих выбор средств разработки:

- возможность создания графического интерфейса;

- поддержка объектно-ориентированного программирования;

- высокая скорость выполнения программного кода;

- наличие библиотек для работы с изображениями.

Первоочередной задачей является выбор языка программирования. Среди большого множества современных языков существует разделение на две группы - компилируемые и интерпретируемые, различающиеся методом исполнения кода. Для разрабатываемого ПС выбрана группа компилируемых языков, так как производительность программы находится в числе важнейших требований. Также язык должен быть объектно-ориентированным, что приводит к ограниченной группе популярных языков: C++, C#, Java, Visual Basic. Java использует особую среду JVM, позволяющую программам выполняться на машинах разных конфигураций, однако требует дополнительных вычислительных затрат, что делает этот язык нежелательным. C++, C# и Visual Basic объединены тем, что могут быть реализованы на одной платформе – Microsoft .NET. Она характеризуется высокой степенью интеграции различных технологий с помощью единого байт кода CIL (Common Intermediate Language) и библиотек DLL (Dynamic Link Library). Так, например, библиотека, написанная на языке Visual C++, может быть использована при написании программ на C# и Visual Basic. Это обеспечивает большую общую базу компонентов для повторного использования. Из трех языков C# наиболее прост в использовании и обеспечивает быструю реализацию объектно-ориентированных архитектур. По производительности он несколько уступает C++, однако этот недостаток может быть преодолен путем использования высокопроизводительных библиотек.

Еще одной характерной особенностью платформы .NET является привязанность к единой интегрированной среде разработки – Microsoft Visual Studio. Помимо редактора исходного кода, она предоставляет широкие возможности для построения схем, анализа кода, управления версиями, визуального программирования и многих других современных методик.

В стандартном пакете библиотек и фреймворков платформы .NET существует один, приспособленный для создания оконных приложений. Windows Forms представляет собой событийно-ориентированный фреймворк. Большая часть времени своей работы приложение ждет действий пользователя, предоставляя для этого графический интерфейс – «форму». В результате какого-либо действия – «события» в терминах фреймворка – исполняется ответственный за его обработку код. Windows Forms предлагает эффективные средства визуального программирования, позволяющие реализовывать графический оконный интерфейс за короткое время с написанием минимального количества кода (для обработки событий).

Платформа .NET имеет ряд встроенных библиотек для работы с графическими данными, однако они ориентированы на удобство и универсальность использования, и поэтому являются недостаточно производительными для обработки больших объемов данных. Для этих целей существует популярная платформонезависимая библиотека OpenCV (Open source Computer Vision). Она состоит из наборов высокопроизводительных функций для обработки изображений в реальном времени. Библиотека написана на языке C++, однако посредством оберточных интерфейсов доступна для языков C#, Perl, Ch, Ruby, Java и Python. Подобной оберткой в случае C# является Emgu CV. Так, через высокоуровневый интерфейс осуществляется доступ к эффективным функциям для обработки графических и статистических (что важно для разрабатываемого ПС) данных.

**4.2** Описание компонентов и классов программы

На аналитической модели приложения, представленной ранее на рисунке 3.1, видны основные элементы архитектуры и их зависимости друг от друга. Элемент «Приложение» - центр всех зависимостей, наиболее специфичный компонент программного средства. Исходящие из него зависимости имеют звездообразную топологию и почти не пересекаются. Это отражает уровни изоляции разных компонентов программы. При эффективном управлении зависимостями в архитектуре, ее составные компоненты могут быть реализованы в виде модулей и библиотек, взаимозаменяемых и используемых повторно. В данной архитектуре можно четко выделить несколько изолированных групп компонентов:

- компонент, отвечающий за синтез панорамы из набора изображений;

- компонент, управляющий файлами;

- окно приложения, обеспечивающее основной графический интерфейс;

- графический редактор;

- компонент, управляющий настройками программы.

Далее они будут рассмотрены подробно.

**4.2.1** Компонент синтеза панорамы

Компонент Panoramas отвечает за реализацию главной функции программы – синтез панорамы как цельного представления множества изображений. Следовательно, входные данные – изображения в общем представлении (пути к файлам, объекты стандартных классов), выходные – одно изображение, также в общепонятном представлении. Также необходим элемент управления, с помощью которого приложению (и пользователю) будет доступен просмотр и изменение настроек генерации результата.

Неоднородность содержания компонента и сложные связи между его модулями говорят о целесообразности применения приема проектирования, известного как «Фасад». Он заключается в создании класса, полностью обеспечивающего интерфейс между компонентом и приложением. В качестве фасада реализован класс Stitcher. Он имеет два конструктора, принимающих либо список имен файлов, которые впоследствии будут загружены объектом класса, либо массив изображений в виде класса стандартного класса Bitmap, а также их имен (для удобства идентификации). Передача непосредственно изображений позволяет избежать потерь времени, требуемого на загрузку и обработку данных с жесткого диска. Члены класса Stitcher:

- Конструкторы. Оба конструктора принимают списки неограниченной длины, однако не меньшей 2. Генерация панорамы из одного изображения не имеет смысла, так как оно уже несет в себе всю доступную информацию. В алгоритме работы программы (подраздел 3.3.1) в процессе синтеза панорамы были определены два четких этапа – анализ сегментов и собственно генерация результата. Вынесение процедуры анализа в интерфейс класса Stitcher усложнило бы его, т.к. назначение и значение метода, его необходимость для последующего вызова StitchAll, являются неочевидными. Выполнение анализа в методе StitchAll противоречит алгоритму работы программы и концентрирует временные затраты в вызове одного метода. Поэтому решено выполнять анализ сразу при инициализации объекта Stitcher и получении им исходных изображений.

- Метод StitchAll выполняет генерацию панорамы. Метод не принимает аргументов, возвращает панораму в самом общем представлении, в виде объекта Bitmap.

- Метод MatchBetween обеспечивает доступ к результатам анализа исходных изображений. Он принимает два имени файла и возвращает экземпляр реализации специального открытого интерфейса IImagesMatch. При инициализации объект класса Stitcher получает имена файлов всех исходных изображений и, следовательно, может в дальнейшем идентифицировать их. MatchBetween осуществляет поиск полученной в результате анализа связи между указанными изображениями.

Интерфейс IImagesMatch определяет, какими параметрами генерации может управлять пользователь. Он должен обеспечивать минимальный необходимый поток информации, не нарушающий правил изоляции компонента Panoramas. Члены интерфейса:

- Метод ToImage. Первоочередным способом отображения информации и связи между сегментами панорамы решено использовать графическое представление. Оно позволяет ограничить количество доступных параметров и одновременно представить их совокупно в наглядном виде. Ввиду специфики используемых методов анализа сегментов, предполагается, что изображение будет содержать информацию об особых точках и связях между ними.

- Свойство CurrentPercent. Отвечает за количество используемых пар особых точек, в порядке убывания их схожести. Значения свойства могут быть в пределах от 0 до 100, независимо от реального количества точек. Таким образом интерфейс ограничивает ненужную выдачу информации внутреннего пользования. Начальное значение CurrentPercent будет отображать автоматически определенное программой количество схожих пар точек.

- Флаг Active. Отвечает за факт использования связи сегментов при генерации панорамы. Пользователь может выключать его, если считает, что сегменты несовместимы друг с другом.

Класс Segment - Один из ключевых классов компонента Panoramas. Объекты класса Segment недоступны через интерфейс модуля. Определенные как элемент типа «Сущность» (подраздел 3.1), они служат для хранения и управления доступом к данным. Так как Segment является главным представлением исходного изображения, он содержит его оригинальное представление Bitmap, а также имя Filename. Помимо этого, каждый сегмент в результате генерации панорамы должен иметь определенную позицию на ней или, в терминах используемой методологии, трансформацию по отношению к глобальному изображению панорамы. Эта информация представлена свойством Transformation. Данное свойство – единственное в классе, которое может быть изменено (в результате перенастройки связей между сегментами). Этой цели служит метод ResetTransformation – безопасный способ вернуть сегмент в свое исходное положение.

Класс Transformation описывает отношение трансформации между двумя изображениями. Он отвечает д