**3 Проектирование программного средства**

**3.1** Разработка архитектуры

Архитектура программного средства – его высокоуровневое структурное описание, определяющее роли основных компонентов системы и связи между ними. Она должна обеспечивать эффективное решение по реализации требований к системе, оптимальным образом отражать варианты использования.

**3.1.1** Среда работы программного средства

Фундаментальные требования к разрабатываемому программному средству были поставлены в разделе 2.1. Из них следует, что ключевыми операциями программного средства будут манипулирование и обработка графической информации. У системы четко определены входные и выходные данные - изображения, причем первые почти однозначно определяют последние. Эта связь дает возможность сделать систему в высокой степени замкнутой, минимизировать ее внешние зависимости, приблизив к концепции «черного ящика». Данный подход тем более целесообразен, если процесс работы ПС имеет комплексный характер. Так, генерация панорамы состоит из ряда этапов и подэтапов обработки данных различного характера.

Однако, полноценный «черный ящик» в данном случае неприменим, так как пользователь должен иметь возможность управлять и вносить изменения в процесс работы ПС. Необходимо предоставлять ему информацию о процессе наиболее оптимальным образом – графическим (ввиду характера работы системы). В современных операционных системах наиболее распространенным средством ввода и вывода информации являются окна – модульные элементы интерфейса, объединенные общей системой управления. Для разработки приложений, использующих собственные окна, существует множество эффективных программных решений. Это говорит о целесообразности разработки программного средства в виде оконного приложения.

Среди требований присутствуют работы с файлами изображений, что означает взаимодействие с файловой системой действующего компьютера. К счастью, все современные платформы для разработки имеют средства абстрагирования подобных взаимодействий, а также их оптимизации. К последним относится использование системных диалоговых окон для удобной работы пользователя с файлами. Это еще один аргумент в пользу оконного приложения.

**3.1.2** Программная архитектура

Процесс генерации панорамы состоит из ряда сложных процедур анализа и преобразования данных различного вида. Это означает наличие высокого риска создания недостаточно гибкой, хрупкой архитектуры, при которой внесение любого изменения в общую структуру может приводить к непредсказуемым последствиям. Поэтому ключевой характеристикой архитектуры должна быть простота, выражаемая в легкости, с которой разработчик сможет ориентироваться в существующей системе и абстрагировать отдельные ее части.

Элементарным приемом для упрощения сложных программных решений является принцип модульности – разделения кода на цельные компоненты. Такой прием позволяет, по возможности, изолировать несвязные фрагменты программы друг от друга, что дает множество преимуществ, среди которых:

- разграничение ответственности;

- возможность параллельной работы нескольких команд разработчиков;

- возможность переиспользования кода;

- простота тестирования.

Разграничение ответственности требует особого способа разбиения кода на модули. Если за некоторый набор функций отвечает четко определенный модуль, работающая над этими функциями команда разработчиков не будет конфликтовать с другими, отвечающими за другие функции. Также ограниченность набора функций модуля означает ограниченный набор способов его использования, и простоту тестирования. Наконец, модульность подразумевает внутреннее разбиение крупных компонент на более мелкие, с отношениями включения. Если схожий код требуется во многих местах, он может быть вынесен в единый модуль, с общей реализацией для всех вариантов использования. Это сокращает общее количество программного кода и избавляет от ошибок при его модификациях.

Модульность упрощает программирование в пределах малого модуля, однако с продвижением вверх по иерархии существенно возрастает уровень ответственности. Современным решением для эффективного абстрагирования и делегирования ответственностей является объектно-ориентированное программирование. Оно основано на представлении программы в виде набора объектов – структур, содержащих информацию и определяющих некоторые операции (методы). Объекты абстрагируются в виде классов, интерфейсов и модулей. Они ссылаются друг на друга и могут составлять иерархии неограниченной сложности. Отношение между двумя классами, когда изменение одного приводит к неизбежному изменению другого, называется зависимостью. Одним из главных достоинств объектно-ориентированного дизайна является то, что он выявляет подобные зависимости и позволяет управлять ими – сокращать или оборачивать вспять. Этой цели служит принцип полиморфизма – использование разных сущностей с помощью одного интерфейса. Использование интерфейсов также означает ограничение взаимодействий с объектом – наличие у него индивидуального пространства для хранения и использования кода (принцип инкапсуляции). Область влияния этого скрытого кода четко определена, что сильно упрощает отладку программы.

Помимо всего вышеназванного, очевидным достоинством объектно-ориентированной архитектуры является ее способность отражать предметную область. Логично организовывать функции и информацию в классы сообразно сущностям, упомянутым в вариантах использования – будь то изображение, панорама или сам пользователь. Оперирование программными абстракциями, имеющими реальные аналоги, облегчает понимание кода и делает разработку отчасти интуитивной. Не все классы в приложении описывают сущности из предметной области. По мере снижения уровня абстракции появляются классы, имеющие специфические роли уже в терминах конкретной программы, платформы разработки или операционной системы.

Существуют устоявшиеся модели распределения и взаимодействия ролей классов в приложении. Использовать их целесообразно при разработке больших программ, при проектировке которых велик риск создания неэффективной архитектуры. Одна из таких моделей – EBC (Entity-Boundary- Control, [X]). Она определяет три категории классов:

- сущности (entity);

- граничные классы (boundary);

- управляющие классы (control).

Сущности – классы, предназначенные для хранения информации, чаще всего относящиеся к предметной области. Граничные классы являются интерфейсом всего приложения, они взаимодействуют с пользователем (или другим актером). Классы управления определяют внутреннюю работу системы по генерации выходных данных на основании входных, полученных из граничных классов. Фактически, классы управления являются непосредственными реализаторами вариантом использования ПС.

Для проектируемого приложения в разделе 2.3 разработана модель предметной области. Она будет использована для конкретизации категорий классов. Так, ключевое понятие – «Панорама» - представляет собой наиболее очевидный кандидат на роль сущности (Panorama). «Сегменты», из которых она состоит, являются контейнерами графической информации – это также сущность в приложении (Segment). «Генератор» - активный функциональный компонент, следовательно, это должен быть класс управления (PanoramaBuilder). «Настройки», которыми он пользуется – еще один контейнер (StitcherSettings). «Проекция панорамы» - панорама, являющаяся простым изображением. Платформа разработки предоставит класс, который заменит эту сущность в приложении. Последнее понятие из модели - «Пользователь». Функции не предполагают хранения какой-либо информации о пользователе. Это понятие отображает инициативную сторону – агента, внешнего по отношению к программному средству. Подобный класс не нужен в приложении.

Кандидаты на роль граничных классов были описаны в разделе 3.1.1 – это форма и диалог для управления файлами.

Функции по загрузке/сохранению файлов, запуску генератора и редактирования изображений, будут реализованы классами управления – соответственно, FileManager, ImageStitcher и ImageEditor. Функция панорамы «представляется в виде проекции» - класс управления PanoramaPresenter.

Все разработанные классы и их зависимости представлены на диаграмме на рисунке 3.1.



Рисунок 3.1 – Аналитическая модель программного средства

**3.2** Взаимодействие пользователя с ПС

Ввиду вышеупомянутой сильной зависимости выходных данных (панорамы) от входных (сегментов), возможность вмешательства в работу может быть излишней в тех случаях, когда система самостоятельно вырабатывает оптимальный результат. Поэтому следует минимизировать необходимость любого вмешательства и предоставить пользователю предварительный результат. В случае неудовлетворительной оценки он должен иметь возможность изменить результат непосредственно или определить этап, на котором дефект может быть устранен, внести изменения в ход этого этапа и оценить их влияние. В случае, если система может самостоятельно оценить качество генерируемой панорамы как неприемлемое, от пользователя изначально потребуется выполнение некоторых корректирующих действий.

**3.3** Проектирование пользовательского интерфейса

**3.4** Разработка основных алгоритмов программы