# Содержание

[Содержание 6](#_Toc198558756)

[Введение 9](#_Toc198558757)

[1 Теоретические и методологические основы работы с эмбоссерами и сопутствующего программного обеспечения 12](#_Toc198558758)

[1.1 Анализ схемы работы и схемы взаимодействия с эмбоссером MATICA 12](#_Toc198558759)

[1.1.1 История возникновения эмбоссеров 12](#_Toc198558760)

[1.2 Изучение программной платформы и поддерживаемого приложения 14](#_Toc198558761)

[1.3 Анализ существующего драйвера 17](#_Toc198558762)

[1.4 Постановка целей и задач 18](#_Toc198558763)

[1.5 Выводы 18](#_Toc198558764)

[2 Проектные решения для разработки драйвера прямого управления эмбоссером matica средствами FloraWare 19](#_Toc198558765)

[2.1 Описание функциональных требований, предъявляемых к разрабатываемому драйверу 19](#_Toc198558766)

[2.1.1. Подключение и Инициализация 19](#_Toc198558767)

[2.1.2 Управление Картой 19](#_Toc198558768)

[2.1.5. Общий Статус и Управление Ошибками 20](#_Toc198558769)

[2.1.6. Конфигурация и Настройки 20](#_Toc198558770)

[2.1.7. Производительность 20](#_Toc198558771)

[2.2 Функциональная схема разрабатываемого драйвера 21](#_Toc198558772)

[2.2.1. Общее описание функциональной схемы 21](#_Toc198558773)

[2.2.2. Описание компонентов функциональной схемы (и элементы на схеме) 21](#_Toc198558774)

[2.2.3. Поток данных и управления 23](#_Toc198558775)

[2.2.4. Альтернативные сценарии 24](#_Toc198558776)

[2.2.5. Заключение 24](#_Toc198558777)

[2.3 Обоснование выбора инструментальных средств разработки 24](#_Toc198558778)

[2.3.1 Описние использованных инструментальных средств 24](#_Toc198558779)

[2.3.2 Заключение 28](#_Toc198558780)

[2.4 Проектирование пользовательского интерфейса 29](#_Toc198558781)

[2.4.1. Общие принципы проектирования интерфейса 29](#_Toc198558782)

[2.4.2. Реализация поддержки IP-адреса 29](#_Toc198558783)

[2.4.3. Визуализация интерфейса 30](#_Toc198558784)

[2.4.4. Обработка ошибок в UI 31](#_Toc198558785)

[2.4.5. Заключение 31](#_Toc198558786)

[2.5 Выводы 31](#_Toc198558787)

[3 Описание и порядок работы с разработанным программным обеспечением 33](#_Toc198558788)

[3.1. Архитектура драйвера (Краткий обзор) 33](#_Toc198558789)

[3.2. Описание API (Интерфейса программирования) 33](#_Toc198558790)

[3.2.1. Обзор классов и функций API 33](#_Toc198558791)

[3.2.2. Подробное описание наиболее важных функций 35](#_Toc198558792)

[3.3. Процесс установки и настройки драйвера 36](#_Toc198558793)

[3.3.1. Системные требования: 36](#_Toc198558794)

[3.3.2. Порядок установки: 36](#_Toc198558795)

[3.3.3. Настройка драйвера: 37](#_Toc198558796)

[3.4. Примеры использования драйвера 37](#_Toc198558797)

[3.4.1. Установление соединения и получение статуса: 37](#_Toc198558798)

[3.4.2. Загрузка карты и эмбоссирование символа: 38](#_Toc198558799)

[3.5. Обработка ошибок 39](#_Toc198558800)

[3.6. Заключение 39](#_Toc198558801)

[3.1 Качественный анализ разработанного драйвера в тестовой среде 39](#_Toc198558802)

[1.2 Передача драйвера клиентам для тестирования 40](#_Toc198558803)

[2.3.6. Дополнительные инструменты 40](#_Toc198558804)

[2.1 Анализ имеющихся средств разработки драйверов 40](#_Toc198558805)

[2.2 Разработка драйвера взаимодействия с эмбоссером 42](#_Toc198558806)

[2.3 Подключение драйвера в приложение 42](#_Toc198558807)

[Заключение 43](#_Toc198558808)

[Список используемой литературы 44](#_Toc198558809)

[Приложения 45](#_Toc198558810)

1.4 Постановка целей и задач

1.5 Выводы по главе

2 Проектное решение разработки драйвера

2.1 Описание функциональных требований

2.2 Функциональная схема разрабатываемого приложения + основные алгоритмы

2.3 Обоснование выбора инструментальных средств разработки

2.4 Проектирование пользовательского интерфейса

2.5 Выводы по главе

3 Описание и порядок работы с разработанным программным обеспечением + Результаты тестирования

# Введение

Современный финансовый сектор, характеризующийся динамичным развитием и непрерывной гонкой за лидерство, требует от банков постоянного совершенствования операционной деятельности и оптимизации бизнес-процессов. В условиях жесткой конкуренции, где качество обслуживания и скорость предоставления услуг становятся ключевыми факторами успеха, банки стремятся минимизировать издержки, повысить производительность и, что самое главное, удовлетворить растущие потребности клиентов. Одним из важнейших элементов в цепочке оказания банковских услуг является процесс выпуска пластиковых карт, который включает в себя множество этапов, от разработки дизайна до персонализации и выдачи готового продукта клиенту.

На этапе персонализации, когда на карту наносится индивидуальная информация, ключевую роль играет эмбоссер – специализированное устройство, предназначенное для рельефного нанесения символов, таких как имя держателя карты, номер карты, срок действия и т.д. Эмбоссирование является неотъемлемой частью процесса выпуска карт и обеспечивает их уникальность и безопасность, а также позволяет проводить транзакции в тех терминалах, которые не поддерживают электронное считывание. Несмотря на появление более современных технологий персонализации, таких как печать, эмбоссирование по-прежнему широко используется, особенно в сегменте премиальных карт, где рельефное нанесение информации подчеркивает их статусность и обеспечивает тактильные ощущения при использовании. Таким образом, от эффективности работы эмбоссеров напрямую зависит не только пропускная способность всего производственного цикла, но и скорость предоставления карт клиентам, что, в свою очередь, оказывает непосредственное влияние на удовлетворенность клиентов и репутацию банка.

Однако, на практике, работа эмбоссеров не всегда соответствует требуемым стандартам эффективности. Низкая производительность, частые сбои в работе, необходимость проведения регулярного технического обслуживания и замены расходных материалов являются серьезными препятствиями, препятствующими эффективному процессу персонализации. Кроме того, недостаточное внимание к технологическим аспектам работы эмбоссеров, а также устаревшие методы их эксплуатации могут приводить к увеличению времени обработки карт, повышению затрат и снижению качества готовой продукции. Учитывая все вышесказанное, исследование путей повышения эффективности работы эмбоссеров для банковских карт является актуальной и практически значимой задачей, решение которой позволит банкам оптимизировать свои операционные процессы, повысить качество предоставляемых услуг и укрепить свои позиции на рынке.

Российская компания «Compass Plus» является лидером в IT-сфере банковского обслуживания. Она предоставляет банкам решения для управления банкоматами и терминалами самообслуживания, фрод-менеджмента, обработки цифровых платежей, торгового эквайринга и т.д. Также, немаловажной отраслью деятельности компании является поддержка эмиссии и персонализации карточных продуктов, которая включает в себя эмбоссирование физической карты. Так как «Compass Plus» охватывает большую долю мирового рынка банковской IT-сферы, для нее необходимо, чтобы выпускаемое ПО работало быстро, стабильно и поддерживало работу с самым современным оборудованием.

Компания «Compass Plus» работает с многими крупными производителями эмбоссеров по всему миру, получая от них документацию для поддержки работы эмбоссеров, используемых клиентами. Среди таких компаний, итальянская компания MATICA является одним из крупнейших производителей эмбоссеров и на данный момент занимает значимую долю рынка.

Целью данной дипломной работы является поддержка персонализации физической карты с помощью эмбоссера компании MATICA через программный продукт компании «Compass Plus», используемый для управления процессом персонализации и эмбоссирования, TranzWare CardFactory.

Для реализации цели необходимо решение следующих задач:

- Выявить схему работы эмбоссера, а также схему его взаимодействия с ЭВМ, используемыми для его управления, изучить возможности программной платформы FloraWare, изучить способы взаимодействия программного продукта TranzWare CardFactory с подключенными к рабочей станции устройствами, раскрыть структуру TranzWare CardFactory, Провести анализ работы существующего модуля управления эмбоссером MATICA;

- Разработать новый модуль для TranzWare CardFactory, который будет осуществлять прямое управление эмбоссером с большей эффективностью;

- Дать оценку эффективности разработанного модуля в сравнении с предыдущей версией;

Объектами исследования являются процесс эмбоссирования, а также возможности программного продукта TranzWare CardFactory.

Предметами исследования являются анализ работы эмбоссера компании MATICA, программная платформа FloraWare и модуль управления эмбоссерами, являющийся частью TranzWare CardFactory.

Данная дипломная работа состоит из введения, трех глав, заключения и списка литературы. Первая глава посвящена теоретическим основам процесса эмбоссирования и анализу существующих технологий. Вторая глава содержит анализ текущего состояния и выявление проблемных областей в работе эмбоссера. В третьей главе проводится разбор процесса разработки модуля, практические рекомендации по повышению эффективности работы эмбоссеров, а также экономическое обоснование данных рекомендаций. В заключении представлены основные выводы и результаты исследования.

# 1 Теоретические и методологические основы работы с эмбоссерами и сопутствующего программного обеспечения

## Анализ схемы работы и схемы взаимодействия с эмбоссером MATICA

### 1.1.1 История возникновения эмбоссеров

Перед тем как описывать непосредственно схему работы эмбоссера, необходимо дать краткое введение в суть процесса эмбоссирования. Следует подчеркнуть, что это механический процесс, в ходе которого на пластиковой карте создается рельефное изображение – выступы или углубления, формирующие текст, цифры или символы. Этот процесс, в отличие от печати, обеспечивает тактильную идентификацию карты и защиту от подделок.

Вся суть работы любого эмбоссера состоит из нескольких этапов, по прохождению которых карта приобретает все присущие ей элементы:

- Загрузка карты;

- Персонализация;

- Эмбоссирование;

Для лучшего понимания работы эмоссера рассмотрим приведенные выше этапы подробнее.

Загрузка карты представляет из себя введение карты в эмбоссер для дальнейшей обработки. Данный этап может быть выполнен как вручную, так и с помощью специального механизма подачи карт, который использует ролики и захваты для переноса карт во внутреннюю часть эмбоссера. После этапа загрузки карты с помощью механизма поступают внутрь специального устройства - кардридера, который зажимает карту. Данное устройство имеет несколько подпружиненных контактов, которые соединяются с соответствующими контактами на карте. Целью кардридера является предоставление интерфейса взаимодействия с картой во время процесса персонализации, а именно возможность записи и считывания информации с чипа на карте. После поступления в кардридер карта проходит процесс персонализации, во время которого на карту записывается вся необходимая банку для работы этой самой карты информация. Далее карта вынимется из кардридера и поступает в эмбоссер, который в свою очередь наносит рисунок на карту и печатает необходимые символы на поверхностях карты с помощью выжигания или выдавливания. Выполнение всех операций производится ЭВМ, подключенной к данному эмбоссеру. После всех вышеперечисленных операций карта считается готовой к эксплуатации и выдается банком клиенту.

Для выяснения схемы взаимодействия эмбоссера MATICA с управляющим приложением, а также выяснения схемы и алгоритма его работы использовалась внутренняя документация компании-производителя. Детально проанализировав схему взаимодействия эмбоссера с внешним ПО, было выявлено несколько ключевых элементов:

1. При работе с эмбоссером, он должен находится в одной сети с рабочей станцией, следовательно, кабель Ethernet, идущий от эмбоссера, подключается через маршрутизатор или же напрямую к рабочей станции
2. Передача команд и информации эмбоссеру происходит по протоколу TCP/IP
3. Все команды и информация передаются эмбоссеру в виде кода, написанного на скриптовом языке программирования TCPscript
4. Если к эмбоссеру подключено несколько станций для записи банковских карт, то он может работать в многопоточном режиме

Углубляясь в значение представленных выше ключевых элементов работы с эмбоссером, а также руководствуясь соответствующей документацией, оказывается, что работа с эмбоссером представляет из себя достаточно простую последовательность действий. Во-первых, происходит подключение эмбоссера к рабочей станции, в большинстве случаев это ПК, на котором имеется необходимое ПО, через кабель Ethernet одним из представленных выше способов. Далее по данному сетевому подключению по протоколу TCP/IP команды и данные в виде непрерывной строки символов с кодом на языке TCPscript передаются эмбоссеру для интерпретации и выполнения. Стоит упомянуть, что скриптовый язык программирования TCPscript имеет C-подобный синтаксис и был разработан компанией «MATICA» специально для общения с эмбоссерами собственного производства. После передачи команды эмбоссеру он интерпретирует и выполняет ее и после этого отправляет ответ о выполненном действии по тому же каналу подключения. Последнее, что стоит упомянуть, так это возможность работы в многопоточном режиме, она осуществляется подключением к эмбоссеру нескольких кардридеров, каждый из которых будет иметь свой собственный IP адрес, по которому к нему можно будет обращаться. IP адрес каждого отдельного кардридера можно определить по базовому адресу, который прописан в документации и является общим для всех эмбоссеров MATICA, адрес первого подключенного ридера является базовым адресом, адрес второго – базовый адрес + 1 и т.д.

Кроме вышеперечисленных ключевых элементов в документации также была представлена схема работы эмбоссера и схема его взаимодействия с управляющим ПО на рабочей станции. Так как все эмбоссеры работают схожим образом, схема работы нас интересует не так сильно, как схема его взаимодействия, поэтому разберем ее подробнее.

Схема взаимодействия эмбоссера с ПО, предоставляемым команией “Compass Plus”:

## 1.2 Изучение программной платформы и поддерживаемого приложения

В первой части данной главы были рассмотрены теоретические основы процесса эмбоссирования и подробно описана схема работы эмбоссера. Мы определили ключевые механизмы и этапы, которые необходимо учитывать при разработке программного обеспечения для управления данным оборудованием. Теперь, когда мы имеем четкое представление о физических принципах работы эмбоссера, нам необходимо перейти к анализу существующего программного обеспечения, используемого для управления этими устройствами.

Как правило, эмбоссеры не работают автономно и требуют программного обеспечения, которое выполняет роль посредника между пользовательским интерфейсом (например, банковской системой) и самим оборудованием. Такое программное обеспечение обычно представлено в виде драйвера, который обеспечивает необходимый интерфейс для взаимодействия с аппаратным обеспечением. Существующие драйверы могут быть разработаны производителями эмбоссеров или сторонними компаниями, и они могут отличаться по архитектуре, функциональности и производительности.

Итальянская компания MATICA поставляет драйверы и приложения для управления в комплекте с эмбоссерами, но компания «Compass Plus» имеет собственное приложение, осуществляющее работу с эмбоссером. Нас интересует драйвер, ранее разработанный бюро продуктов персонализации внутри компании. Данный драйвер был разработан как часть модуля Embosser Driver приложения TranzWare CardFactory. Данное приложение разработано на основе программной платформы FloraWare, которая предоставляет широкие возможности для разработки программных продуктов разной сложности.

Чтобы ясно понять технологическую составляющую работы, обратим свое внимание на центральное ПО, на основе которого будет проводится вся дальнейшая разработка – программная платформа FloraWare. Вопрос стоит в надобности данной платформы, так как для разработки приложений могут использоваться и более распространенные и современные средства разработки.

Попытки создания полноценной объектной машины и замены текстового представления программы предпринимались неоднократно, однако все они ограничивались полумерами. Это можно объяснить несколькими причинами. Технология, меняющая что-то кардинально, требует отбрасывания существующих программных компонентов, созданных по другим технологиям. Поэтому она обязана предложить весь спектр инструментальных средств, либо как запасной вариант, предоставить инструменты поддержки других технологий, и то и другое требует больших усилий. С другой стороны ОО языки программирования создавались с учетом универсальности объектов и полного контроля над ними со стороны пользовательской программы. Даже если технология предполагает наличие некой виртуальной машины, назвать ее объектной можно только с большой натяжкой, на откуп ей в лучшем случае отдается сборка мусора. Тогда как наибольшие преимущества объектной машины сказываются при максимальном ее контроле над объектами. Попытки заменить текстовое представление алгоритма, в большинстве случаев, сводятся к генерации текста части программы на основе ее графического представления с последующей трансляцией, интересным же является полное исключение теста программы. Избавиться от перечисленных недостатков можно, отказавшись от архитектуры машины с потоком команд и линейной памятью, и перейдя к архитектуре с древовидной объектной памятью, с управлением объектами собственно объектной машиной.

На примере технологии программирования FloraWare сделана попытка реализации полноценной объектной машины как основы реализации объектного метода. С практической точки зрения, целью создания FloraWare было построение простой и удобной объектно-ориентированной среды разработки и исполнения приложений для больших программных систем со средствами коллективной разработки и удаленной отладки приложений.

Объектная машина - основа реализации объектного метода, должна иметь инструменты поддержки неотъемлемых составляющих объекта: *состояния*, *поведения* и *идентичности*, а также инструменты реализации отношений между объектами: *связей* и *агрегации*. В объектной машине *FloraWare* в качестве форматов данных выступают объекты, а алгоритм реализуется передачей сообщений между объектами и их реакцией на эти сообщения. Архитектурно объектная машина *FloraWare* может быть представлена в виде нескольких уровней.

На самом нижнем уровне реализован аппарат работы со свойствами объектов, которые относятся к четырем базовым типам: целое, действительное, строка и указатель на объект, эти свойства могут отличаться набором индексов базовых типов. Свойства объекта это инструмент поддержки его *состояния*.

Наличие объектов-*контейнеров*, способных содержать внутри себя другие объекты, позволяет строить сколь угодно сложные объекты. Обязательный объект - *корневой контейнер*, в который по мере необходимости добавляются другие объекты, реализует память объектной машины представляющую собой дерево, в котором находятся все объекты, существующие в данный момент времени. Добавление всем объектам выделенного строкового свойства *Имя* позволяет *идентифицировать* объекты и осуществлять навигацию по дереву объектов. Составное свойство *Ссылка на объект*, состоящее из относительного пути до объекта и указателя на объект, дает возможность организовывать сколь угодно сложные связи между объектами. Таким образом, древовидная структура данных является основой реализации отношения *агрегация* между объектами, а с помощью ссылок на объекты устанавливаются отношения типа *связь*.

Второй уровень объектной машины обеспечивает работу объектов встроенных базовых классов, которые освобождают пользователя от проектирования самих этих классов, в его обязанности входит только определение использования их экземпляров. Встроенные классы реализованы в ядре *FloraWare* как обычные классы, созданные с использованием объектно-ориентированного языка программирования, со своими свойствами и методами.

Разобравшись с внутренним устройством FloraWare, изучим разработанное на основе этой платформы приложение – TranzWare CardFactory, которое является основным ПО, разработанным компанией «Compass Plus», реализующим работу c эмбоссерами и кардридерами. Данное приложение позволяет банкам-эмитентам выпускать карточную продукцию используя купленное оборудование, а также осуществлять контроль процесса обработки карты. Для осуществления контроля над полным спектром возможностей обработки карточных продуктов внутри TranzWare CardFactory разработано множество модулей, отвечающих за настройку и реализацию определенных этапов обработки карты. Так как в контексе данной работы нас не интересуют все имеющиеся на данный момент модули, мы рассмотрим только минимальный и достаточный набор модулей для успешной обработки и эмбоссирования карты. Необходимый и минимальный набор модулей для наших целей дожен состоять из:

Анализируя имеющееся приложение для работы с банковскими картами, частью которого и является вышеупомянутый драйвер, было выяснено, что для корректной работы драйвера и его регистрации приложением недостаточно его наличия, необходимо также, чтобы существование драйвера было зарегистрировано в базе данных, используемой приложением для хранения данных о большинстве собственных модулей.

При запуске модуля обработки карты внутри TranzWare CardFactory, при условии, что во время настройки карточного продукта среди эмбоссеров был выбран эмбоссер компании MATICA. Данный драйвер подключится к модулю урпавления эмбоссерами, когда он начнет свою работу. После этого внутри модуля будет запущена функция Start, которая и осуществит общение с эмбоссером и эмбоссирует карту как это было настроено внутри приложения. Во время эмбоссирования будут последовательно запускаться функции Send и Receive, которые непосредственно осуществляют обмен информации с эмбоссером.

## 1.3 Анализ существующего драйвера

Подробное изучение программной платфофрмы и построенного на нем приложения поможет лучше понять уже разработанные драйверы для работы с эмбоссерами других производителей. Анализ данных драйверов поможет лучше понять возможности для разработки подобных драйверов, а также понять как строится структура любого драйвера для взаимодействия с эмбоссерами.

Изучая разработанный на платформе FloraWare драйвер для работы с эмбоссером «Вставьте название здесь», а также руководствуясь внутренней документацией компании по языку программирования F++, являющемся основой разработки ПО на вышеупомянутой технологической платформе, внутри драйвера были определены все элементы работы с эмбоссером, представленные в предыдущей главе. Частью драйвера являются функции: отправки сообщений эмбоссеру, получения ответа от эмбоссера, логирования выполненных действий, проверки подключения, получения информации о записываемой карте и т.д. Технологическая платформа FloraWare предоставляет все необходимые инструменты для реализации данных функций и обеспечения полноценной работы с эмбоссерами.

Данный драйвер является частью модуля Chip Writer Driver и реализует схему взаимодействия с эмбоссером, описанную в первой части данной главы. Он отвечает за взаимодействия с эмбоссером на этапе персонализации карты. Драйвер выступает в роли посредника, который осуществляет запись информации на физическую карту средствами специализированных драйверов для записи определенных типов карточных продуктов внутри TranzWare CardFactory. Карточные драйвера предоставляют всю необходимую для записи на карту информацию и алгоритм записи этой информации, драйвер эмбоссера же является всего лишь интерфейсом взаимодействия, который предоставляет возможность карточному драйверу установить связь с физической картой.

Реализуемый драйвером эмбоссера интерфейс взаимодействия представляется двумя функциями Send и Receive, которые реализуют отправку сообщений на карту и получения ответа с карты соответственно. Во время персонализации карты, эти функции вызываются карточным драйвером и в функцию отправки при вызове передаются данные для отпраки на карту, функция получения ответа же используется только для проверки положительного ответа об успешно завершенной операции записи с карты.

При запуске модуля обработки карты внутри TranzWare CardFactory, при условии, что во время настройки карточного продукта среди эмбоссеров был выбран необходимый эмбоссер. Данный драйвер подключится к модулю управления эмбоссерами, когда он начнет свою работу. После этого внутри модуля будет запущена функция Start, внутри которой реализован алгоритм записи карты, который как раз таки использует функции Send и Receive при возникновении необходимости записи информации на карту. Каждый запуск данных функций будет логироваться с помощью специальной функции Log, которая будет записывать логируемое сообщение в специальный буфер или же в текстовый файл, если был выствален соответствующий флаг в интерфейсе запуска обработки карты. Логироваться будет только переданная на карту информация и полученное с карты сообщение.

## 1.4 Постановка целей и задач

Разработка драйвера, подобного представленному выше, представляет из себя долгий и трудоемкий процесс, который предполагает изучения как методов взаимодействия с эмбоссером, так и встроенных методов передачи комманд из программной платформы. Кроме того, необходимо необходимо правильно адаптировать передачу комманд под разрабатываемый эмбоссер, так как изначально в драйвер комманды поступают так, как они были записаны в CardFactory, и задачей драйвера, кроме передачи комманд и принятия сообщений, является также парсинг комманды от CardFactory и правильное построение отправляемой эмбоссеру комманды на основе предоставленных данных. Итак, на основании всего вышеперечисленного можно составить актуальный список целей и зачад для осуществления успешной разработки драйвера для осуществления прямого управления эмбоссером MATICA:

* Создание или копирование шаблона драйвера по примеру существующих драйверов
* Изменение существующих алгоритмов для соответствия методике работы с эмбоссером
* Правка существующего интерфейса для передачи дополнительных параметров в драйвер
* Проверка работоспособности разработанного драйвера в тестовой среде, а также на реальном эмбоссере

## 1.5 Выводы

По итогам данной главы, была изучена и исследована вся теоретическая основа, необходимая для разработки драйвера управления эмбоссером. Была изучена программная платформа, на основе которой строится все приложение, в которое будет включаться разрабатываемый драйвер, а также на основе которой будет разрабатываться данный драйвер.

# 2 Проектные решения для разработки драйвера прямого управления эмбоссером matica средствами FloraWare

## 2.1 Описание функциональных требований, предъявляемых к разрабатываемому драйверу

В данном разделе описываются основные задачи, выполняемые драйвером прямого управления эмбоссером MATICA в соответствии с предъявляемыми техническими требованиями. Функциональные требования служат основой для проектирования, разработки, тестирования и документирования драйвера. Они помогают разработчикам четко понимать, какого результата нужно добиться в конечном итоге, и помогают легче планировать дальнейшую разработку и разбивать весь процесс на отдельные части. Все функциональные требования будут разбиты на категории, в соответствии с частями функционала драйвера, который они описывают. Конечно, функциональные требования будут разрабатываться с учетом разработки драйвера на платформе FloraWare и с учетом того, что он будет являться частью программного продукта TranzWare CardFactory.

Список функциональных требований, предъявляемых к драйверу прямого управления эмбоссером MATICA:

* Требования к подключению и инициализации:
  + Установка соединения: драйвер должен устанавливать соединение с эмбоссером по протоколу TCP/IP. Необходимо, чтобы при этом эмбоссер находился в одной локальной сети с управляющей ЭВМ или же подключен напрямую в один из портов Ethernet управляющей ЭВМ.
  + Поддержка нескольких соединений: драйвер должен поддерживать подключение только к одной управляющей плате и иметь возможность обращаться к любому кардридеру этой платы в соответствии с настройками.
  + Передача сообщений: все сообщения, которые планируются для передачи карте должны посылаться управляющей плате по соответствующему для нужного кардридера IP адресу по протоколу TCP/IP в виде строки символов, в которой будет содержаться команда передачи данных карте, написанная на языке TCPscript.
  + Обработка ошибок соединения: если посланное драйвером сообщение не получило ответа от эмбоссера или же в ответ получена ошибка, то необходимо прекратить обрабоку карты и отобразить сообщение об ошибке в логах обработки или всплывающим окном.
* Требования к управлению картой:
  + Передача команд: разрабатываемый драйвер должен вызываться драйвером записи карты и передавать карте только те данные, которые поступают ему от этого драйвера.
  + Обработка сообщений карты: сообщения, полученные от карты должны передаваться вызывающему драйверу записи карты.
* Требования к конфигурации и настройкам
  + Передача настроек: все настройки должны передаваться в драйвер посредством указания их в интерфейсе модуля ChipWriterDriver.
  + –прописать откуда(из какого файла) драйвер берет настройки
* Требования к производительности
  + *Минимальные задержки:* драйвер должен обеспечивать минимальные задержки при обмене данными с эмбоссером, чтобы максимально повысить скорость работы.
  + *Оптимизация обмена данными:* драйвер должен использовать эффективные алгоритмы для обмена данными с эмбоссером, минимизируя количество передаваемых байтов.

--Перенести в теорию-- Из всех модулей, составляющих эмбоссер, разрабатываемый драйвер осуществляет управление только модулем кардридера, остальные же модули эмбоссера управляются программным модулем *EmbosserDriver* в *CardFactory*. Это будет учтено при выведении функциональных требований к драйверу. ------

Определение функциональных требований это необходимый процесс, который поможет в дальнейших этапах планирования, а также на этапе разработки драйвера. Этот раздел определил полный список функциональных требований, предъявляемых к разрабатываемому драйверу. Данный список будет активно использоваться в дальнейших этапах разработки функциональной схемы драйвера и проектирования интерфейса. В следующих разделах будут рассмотрены нефункциональные требования и ограничения, а также архитектура драйвера.

## 2.2 Функциональная схема разрабатываемого драйвера

Этот раздел представляет функциональную схему разрабатываемого драйвера, построенную в соответствии с ГОСТ 19.701-90, демонстрирующую взаимодействие между его основными компонентами и внешними сущностями, такими как эмбоссер и вызывающее приложение. Функциональная схема обеспечивает визуальное представление архитектуры драйвера и помогает понять поток данных и логику управления.

-- Функциональная схема с блок схемой и описанием каждого элемента схемы и всех связей передач и т.д. --

### 2.2.1. Общее описание функциональной схемы

Функциональная схема (см. Рисунок 2.1) разделена на несколько уровней абстракции:

* **Уровень 1: Вызывающее приложение:** Представляет собой программу, использующую драйвер для управления эмбоссером (например, банковская система персонализации карт). Это клиент драйвера, которым является приложение CardFactory.
* **Уровень 2: Интерфейс драйвера (API):** Предоставляет набор функций и методов, доступных для вызывающего приложения. Этот уровень абстрагирует детали реализации драйвера.
* **Уровень 3: Ядро драйвера:** Содержит основную логику управления эмбоссером, включая управление соединением, обработку команд и ошибок, а также взаимодействие с аппаратным интерфейсом.
* **Уровень 4: Аппаратный интерфейс:** Представляет собой механизм физического соединения с эмбоссером через Ethernet порт.
* **Уровень 5: Эмбоссер:** Физическое устройство, выполняющее операции эмбоссирования, кодирования магнитной полосы и другие функции.

Рисунок 2.1: Функциональная схема разрабатываемого драйвера

### 2.2.2. Описание компонентов функциональной схемы (и элементы на схеме)

На схеме должны быть четко обозначены следующие компоненты, а связи между ними показаны стрелками, отражающими направление потока данных и управления:

* **Вызывающее приложение:**
  + *Ввод:* Посылает команды драйверу через API (например, EmbossCard, LoadCard, GetStatus).
  + *Вывод:* Получает статус эмбоссера, информацию об ошибках, и подтверждение успешного выполнения команд.
* **Интерфейс драйвера (API):**
  + *Функции:*
    - Connect(InterfaceType, Configuration): Устанавливает соединение с эмбоссером.
    - Disconnect(): Завершает соединение с эмбоссером.
    - LoadCard(): Загружает карту в эмбоссер.
    - EjectCard(): Выгружает карту из эмбоссера.
    - EmbossCharacter(Character, Position): Отправляет команду эмбоссирования символа.
    - GetStatus(): Получает статус эмбоссера.
    - Configure(Parameter, Value): Устанавливает параметры конфигурации драйвера.
  + *Описание:* Предоставляет высокоуровневые функции для управления эмбоссером. Обеспечивает абстракцию от аппаратных деталей.
* **Ядро драйвера:**
  + *Компоненты:*
    - **Менеджер соединений:** Управляет установлением и завершением соединения с эмбоссером.
    - **Обработчик команд:** Преобразует команды API в специфичные для эмбоссера команды. Управляет очередью команд.
    - **Драйвер устройства (Device Driver):** ( *зависит от ОС* ) Непосредственно взаимодействует с аппаратным интерфейсом (например, через USB stack или драйвер последовательного порта).
    - **Обработчик ошибок:** Обрабатывает ошибки, полученные от эмбоссера и аппаратного интерфейса, и передает информацию об ошибках вызывающему приложению.
    - **Менеджер конфигурации:** Управляет настройками конфигурации драйвера.
  + *Функции:*
    - Прием и обработка команд от API.
    - Преобразование команд в низкоуровневые команды для эмбоссера.
    - Обмен данными с эмбоссером через аппаратный интерфейс.
    - Управление потоком команд.
    - Обработка ошибок и уведомление вызывающего приложения.
    - Ведение журнала ошибок.
* **Аппаратный интерфейс:**
  + *Интерфейсы:*
    - Ethernet (TCP/IP)
  + *Функции:*
    - Физическая передача данных между драйвером и эмбоссером.
* **Эмбоссер:**
  + *Функции:*
    - Прием команд от драйвера.
    - Выполнение операций эмбоссирования, кодирования магнитной полосы (опционально), и других функций.
    - Отправка информации о статусе и ошибках драйверу.

### 2.2.3. Поток данных и управления

1. Вызывающее приложение отправляет команду драйверу через API.
2. API передает команду ядру драйвера.
3. Обработчик команд преобразует команду в специфичный для эмбоссера формат.
4. Драйвер устройства отправляет низкоуровневую команду эмбоссеру через аппаратный интерфейс.
5. Эмбоссер выполняет команду и возвращает статус драйверу устройства.
6. Ядро драйвера обрабатывает статус и передает информацию вызывающему приложению через API.
7. В случае возникновения ошибки, обработчик ошибок генерирует соответствующее уведомление и записывает информацию в журнал.

### 2.2.4. Альтернативные сценарии

* **Ошибка соединения:** Если не удается установить соединение с эмбоссером, менеджер соединений генерирует ошибку, которая передается вызывающему приложению.
* **Ошибка выполнения команды:** Если эмбоссер возвращает ошибку при выполнении команды, обработчик ошибок обрабатывает ошибку и передает информацию вызывающему приложению.
* **Асинхронные уведомления:** Драйвер может генерировать асинхронные уведомления для вызывающего приложения (например, об окончании ленты), используя механизм обратного вызова.

### 2.2.5. Заключение

Эта функциональная схема обеспечивает четкое представление архитектуры разрабатываемого драйвера. Она демонстрирует взаимодействие между различными компонентами и поток данных и управления. Эта схема будет использована в следующих разделах для более детального описания реализации драйвера. В следующем разделе будут определены нефункциональные требования к драйверу.

## 2.3 Обоснование выбора инструментальных средств разработки

Выбор всех инструментальных средств разработки был продиктован требованиями поставленного технического задания. В данном разделе будет рассмотрен список таких средств, а также рассмотрены свойства, парметры и функционал данных средств, которые непосредственно влияют на разработку и дальнейшее распространение готового программного продукта.

### 2.3.1 Описние использованных инструментальных средств

В качестве ОС для базирования разработки в соответствии с техническими требованиями использовалась система Microsoft Windows. На данной системе разработана вся экосистема FloraWare, так что она является основопологающей при осуществлении любых разработок для FloraWare. Система Windows обладает качествами, которые напрямую повлияют на тестирование и распространение готового драйвера, а именно:

* Широкое распространение: Windows является наиболее распространенной ОС на рынке, что обеспечивает широкую совместимость с различными эмбоссерами и программным обеспечением, которое будет использовать драйвер, в том числе с ПО для работы с эмбоссером MATICA;
* Совместимость с оборудованием: Windows имеет обширную поддержку аппаратного обеспечения, что упрощает интеграцию с различными моделями эмбоссеров;
* Поддержка программной платформы: Windows нативно поддерживает программную платформу FloraWare, на которой производится вся разработка;
* Популярность среди клиентов: подавляющее большинство клиентов CompassPlus, пользующихся программным продуктом CardFactory, используют Windows для взаимодействия с продуктом CardFactory и эмбоссерами.

При возникновении у клиентов необходимости поддержки драйвера на других ОС, присутствует возможность внесения драйвера в версию приложения, разрабатываемую под семейство систем Linux, но в таком случае драйвер придется отдельно вносить и пересобирать на данной системе.

В качестве языка программирования для разработки драйвера использовался язык F++, который является встроенным в программную платформу FloraWare. Данный язык используется для разработки любого программного продукта на базе FloraWare и является единственным языком, выполняющим подобную функцию внутри экосистемы. Весь набор инструментов, присутствующих в языке, построен на основе требований к инструментарию для разработки продуктов, разрабатываемых на основе FloraWare, таких как CardFactory. Любой язык программирования напрямую влияет на процесс разработки и тестирования продуктов, а также способы распространения и F++ не является исключением. Он обладает всеми необходимыми параметрами для осуществления разработки драйвера, а именно:

* *Широкие возможности развития*: язык F++ постоянно расширяется, пополняясь новыми встроенными функциями и объектами. Так как язык полностью поддерживается и дорабатывается компанией CompassPlus, то при возникновении необходимости в том или ином функционале присутствует возможность отправки запроса на добавление нового функционала, после чего данный запрос будет рассмотрен и при одобрении внесен в будущую версию FloraWare,
* *Минимально необходимый функционал*: встроенный в FloraWare язык программирования предоставляет минимально необходимый функционал для разработки поддерживаемых на данной платформе программных продуктов, из-за этого он не является настолько «раздутым», как достаточно популярные на данный момент языки C++, C#, Java, где есть множество вариантов решения одной и той же задачи встроенными средствами языка, а также огромное множество встроенных библиотек и модулей. Язык F++ в свою очередь предлагает более скромный, но достаточный для разработки инструментарий, а также он не разбит на огромное множетсво модулей, все встроенные элементы доступны сразу, без необходимости дополнительного подключения;
* Привычный синтаксис: синтаксис языка F++ разработан как C-подобный, следовательно, разработчики смогут быстрее и без больших усилий приспособиться к данному языку программирования.

При разработке на базе программной платформы FloraWare возможности выбора подходящей среды разработки не так широки, как в системе Windows, других ОС и программных платформах. В данной экосистеме существует единственная доступная встроенная среда разработки Designer. Среда разработки предоставляет разработчику все необходимые инструменты для облегчения и ускорения процесса разработки. Designer разработан с целью служить ключевым средством разработки экосистемы FloraWare и он обладает всеми необходимыми свойствами для полноценной реализации разрабатываемого драйвера, такими как:

* Полная объектная библиотека платформы: среда разработки Designer предоставляет возможности для просмотра полной библиотеки объектов платформы и добавления элементов этой библиотеки в приложение, отчего разработка становится более быстрой и интуитивной;
* Удобный пользовательский интерфейс: Designer предоставляет удобный и интуитивно понятный пользовательский интерфейс для просмотра и редактирования дерева объектов разрабатываемого программного продукта, а также посредством интерфейсных элементов, он предоставляет возможности взаимодействия с этим деревом, такие как копирование и вставка объектов дерева, добавление и удаление объектов, поиск объектов в дереве по различным фильтрам и т.д.
* Инструмент «Палец»: нструмент «Палец» является одним из наиболее полезных инструментов, встроенных в Designer. Он позволяет указать на любой интерфейсный элемент в любом открытом в среде программной платформы приложении и открыть его инициализированное дерево или же показать элемент, на который указал палец в файле объектного хранилища. Данный инструмент позволяет существенно ускорить разработку, так как упрощает процесс поиска в файлах хранилищ и объектных деревьях нужного графического элемента или модуля, а также позволяет ускорить отладку и тестирования разработанных приложений, так как позволяет отрыть инициализированное дерево, в котором можно будет наблюдать текущее состояние объектов и их изменение в реальном времени.

Кроме средств разработки платформа FloraWare закономерно имеет необходимые встроенные средства отладки и тестирования разработанных продуктов, которые комплементируют имеющиеся средства разработки. Таким средством является приложение Debugger. Не нужно упоминать, что наличие таких средств в платформе позволяет облегчить процесс обнаружения проблем в работе модулей и их исправления, а также предоставляет инструменты для наблюдения потока выполнения программы, для обнаружения неэффективных реализаций и неочевидных ошибок. Debugger обладает рядом свойств, которые делают его незамимым при осуществлении любых разработок в экосистеме FloraWare, а именно:

* *Интеграция с Designer*: при условии, что внутри кода какого-либо объекта-функции или объекта-задачи была установлена точка остановки с помощью утилиты Desiner, а также при условии, что запущена утилита Debugger, во время выполнения кода данного объекта при достижении точки остановки Debugger остановит запущенное дерево и предоставит возможность управлять потоком выполнения программы, а также просматривать содержимое локальных переменных и объектного дерева. Таким образом, используя в тандеме утилиты Designer и Debugger, разработчик может добиться высокой производительности своей работы, так как может параллельно пошагово выполнять код в Debugger и просматривать изменения в дереве внутри Designer;
* *Минимальный необходимый интерфейс*: Debugger является самостоятельной утилитой, отделенной от среды разработки платформы, отчего интерфейс и функционал Debugger-а предоставляет только все необходимое для управления потоком выполнения программы, отлавливания точек остановки и просмотра стека памяти проограммы, что помогает разработчику легче приспособиться к работе с утилитой и ускоряет процесс разработки;

Также, в продукте CardFactory присутствует собственный инструемент отладки и тестирования в виде логов обработки карты, которые записываются каждым модулем и логируют каждое их действие и которые будут использоваться для отслеживания точности информации, передаваемой карте через драйвер, а также для отслеживания ответа, принимаемого драйвером от карты на посылаемые данные.

Кроме инструментов для разработки и тестирования, ни один крупный проект по разработке не обходится без системы управления версиями(VCS – version control system). Она помогает командам разработчиков эффективнее вместе работать над одним проектом. С помощью таких систем любой разработчик в команде сможет отслеживать выполненную работу, а также сможет загружать свои изменения в общий репозиторий, причем современные системы управления версиями предупреждают разработчиков о пересекающихся изменениях, что позволяет избежать перетирания изменений. Для разработки программного продукта TranzWare CardFactory используется клиент Tortoise SVN для сервера VCS Apache Subversion. Функционал и характеристики клиента Tortoise SVN напрямую влияют на процесс разработки продукта, а также на дальнейшее внесение готового решения в рабочую версию CardFactory, конкретнее:,

* *Удобный интерфейс*: интерфейс Tortoise SVN предлагает очень удобное взаимодействие с репозиториями. Данная система не предоставляет общий универсальный интерфейс, но предоставляет несколько отдельных окон для выполнения операций, таких как клонирование репозитория, откат к ревизии, просмотр списка ревизий. Также хотелось бы отметить, что интерфейс для просмотра списка ревизий выполнен очень удобно и предоставляет всю необходимую информацию о ревизиях, такую как описание ревизии, номер ревизии и список файлов, что позволяет оперативно получать информацию о ревизиях и производить быстрый поиск по списку;
* *Проработанная интеграция с ОС Windows*: Tortoise SVN очень хорошо интегрируется в ОС Windows, добавляя собственный функционал в контекстное меню папок с репозиториями, позволяя осуществлять взаимодействие с репозиторием через контекстное меню проводника. Также, утилита изменяет пиктограммы папок и файлов, если они находятся под версионным контролем, добавляя к ним небольшое изображение в зависимости от их статуса: соответствует ревизии – зеленая галочка, файл изменен – красный крест, конфликт с ревизией – желтый треугольник с черным восклицательным знаком.

### 2.3.2 Заключение

Набор инструментальных средств разработки является основополагающим фактором для разработки любого программного продукта. Определение такого набора напрямую влияет на скорость разработки, количество допускаемых ошибок и возникаемых багов, масштабируемость проекта, способы распространения проекта и т.д. Инструментальные средства должны быть обязательно определены до начала процесса разработки для того, чтобы четко выбрать самые эффективные для поставленной задачи средства и не останавливать процесс разработки для подбора необходимых инструментов. А иначе возможна такая ситуация, что в процессе разработки проекта будут выявлены недостатки того или иного инструмента и возникнет необходимость перехода на другой. В больших проектах такие переходы могут быть очень трудозатратными, поэтому необходимо по возможности их избегать и четко определять набор инструментов, который будет использоваться на протяжении всей жизни проекта, еще на этапе планирования. Конечно, присутствует возможность оставить уже используемые инструменты, но в таком случае разработчикам придется мириться с их недостатками, что существенно снизит производительность. Представленный в предыдущем разделе инструментарий удовлетворяет все требованиям для осуществления полноценной разработки и тестирования драйвера прямого управления эмбоссером MATICA. Часть представленного инструментария будет использована в следующем разделе для описания процесса проектирования пользовательского интерфейса.

## 2.4 Проектирование пользовательского интерфейса

В данном разделе описывается проектирование пользовательского интерфейса (UI) для модуля ChipWriterDriver. UI предназначен для упрощения настройки и управления драйвером, а также для обеспечения удобства взаимодействия с пользователем. В данном случае, мы сосредоточимся на конкретных изменениях, внесенных в интерфейс для поддержки сетевого подключения к эмбоссеру по IP-адресу.

### 2.4.1. Общие принципы проектирования интерфейса

При проектировании UI для ChipWriterDriver были учтены следующие принципы:

* **Простота и понятность:** Интерфейс должен быть интуитивно понятным и легким в использовании даже для неопытных пользователей.
* **Функциональность:** UI должен предоставлять доступ ко всем основным функциям драйвера, таким как подключение, настройка, выполнение операций и получение статуса.
* **Наглядность:** UI должен визуально отображать текущий статус эмбоссера и состояние выполняемых операций.
* **Настраиваемость:** UI должен позволять пользователю настраивать параметры соединения и другие параметры драйвера.
* **Обратная связь:** UI должен предоставлять пользователю обратную связь о результатах выполнения операций и возникающих ошибках.
* **Соответствие стандартам:** UI должен соответствовать общепринятым стандартам проектирования интерфейсов для используемой операционной системы.

### 2.4.2. Реализация поддержки IP-адреса

В рамках данной дипломной работы было внесено значительное улучшение в UI модуля ChipWriterDriver – добавлена поддержка сетевого подключения к эмбоссеру по IP-адресу. Это потребовало внесения изменений в интерфейс для настройки параметров подключения. Ниже приводится описание этих изменений:

* *Добавление поля ввода IP-адреса:*
  + В UI был добавлен текстовый input field, позволяющий пользователю ввести IP-адрес эмбоссера. Это поле расположено в разделе настроек соединения, в нижней части интерфейса модуля
  + Тип поля: Текстовое поле, принимающее валидные IP-адреса с использованием регулярного выражения
  + Метка: “IP-адрес эмбоссера:”.
  + Поведение: Поле активно только в том случае, если выбрана опция сетевого подключения (см ниже), а также выбран необходимый тип эмбоссера.
* *Добавление флажка (checkbox) выбора режима подключения:*
  + В UI был добавлен флажок (checkbox) с меткой, указывающей на выбор сетевого подключения. Например: “Подключение по IP”.
  + Поведение:
    - Если флажок отмечен (checked), пользователь может ввести IP-адрес эмбоссера в соответствующем поле.
    - Если флажок не отмечен (unchecked), поле ввода IP-адреса становится неактивным, и используется другой режим подключения.
    - По умолчанию флажок не отмечен, и используется стандартный режим подключения через интерфейс USB в зависимости от предыдущих настроек или настроек по умолчанию.
* *Взаимодействие с другими элементами интерфейса:*
  + При изменении состояния флажка (“Подключение по IP”) должно происходить соответствующее изменение активности поля ввода IP-адреса.
  + При выборе сетевого подключения, другие параметры, связанные с подключением через последовательный порт (COM-порт, скорость передачи данных и т.д.) должны быть скрыты или неактивны.
  + UI должен обеспечивать проверку введенного IP-адреса на валидность, прежде чем разрешать подключение. В случае неверного ввода должна отображаться соответствующая ошибка.

### 2.4.3. Визуализация интерфейса

**--Вставить картинку интерфейса--**

UI для модуля ChipWriterDriver предоставляет интерфейс для настройки параметров подключения. Основной раздел настроек подключения содержит следующие элементы:

* Метка: “Подключение по IP:”. Флажок (checkbox). По умолчанию не отмечен.
* Метка: “IP-адрес эмбоссера:”. Текстовое поле, активное только при отмеченном флажке “Подключение по IP”. По умолчанию неактивно.
* Кнопка: “Подключиться”. Запускает процесс подключения к эмбоссеру с использованием выбранных параметров.

### 2.4.4. Обработка ошибок в UI

Интерфейс любого программного продукта должен соответствовать требованиям, необходимым для его удобной и правильной эксплуатации пользователем. Данные требования включают в себя валидацию действий пользователя и выдача соответствующих сообщений об ошибке или какой-либо цветовой или иной идентификации ошибок в действиях пользователя. В процессе определения требований к программному продукту, было выявлено несколько таких требований к пользовательскому интерфейсу:

* UI должен корректно обрабатывать ошибки, возникающие при вводе IP-адреса и при попытке соединения по IP;
* В случае неверного IP-адреса, в UI должно отображаться соответствующее сообщение об ошибке;
* В случае ошибки при подключении, UI должен отображать сообщение об ошибке и, при необходимости, предложить варианты исправления.

### 2.4.5. Заключение

В этом разделе было представлено проектирование пользовательского интерфейса для модуля ChipWriterDriver, включая добавление поля для ввода IP-адреса и флажка для выбора режима подключения по IP. В следующих разделах будут рассмотрены аспекты реализации драйвера и результаты его тестирования.

## 2.5 Выводы

В данной главе были определены основные аспекты разработки драйвера прямого управления эмбоссером, начиная от функциональных требований и заканчивая проектированием пользовательского интерфейса. На основе проведенного анализа можно сделать следующие выводы:

* **Функциональные требования:** Были четко сформулированы функциональные требования к драйверу, охватывающие все необходимые операции для управления эмбоссером, включая подключение, инициализацию, управление картой, эмбоссирование и обработку ошибок. Эти требования послужат основой для реализации драйвера и его тестирования. Особое внимание уделено требованиям к обработке ошибок и безопасности, что обеспечит надежную и безопасную работу системы.
* **Функциональная схема:** Разработанная функциональная схема позволила визуализировать архитектуру драйвера и определить взаимодействие между его основными компонентами. Схема наглядно демонстрирует поток данных и управления, что упрощает понимание логики работы драйвера и его дальнейшую разработку. Разделение на уровни абстракции (вызывающее приложение, API, ядро драйвера, аппаратный интерфейс, эмбоссер) обеспечивает модульность и гибкость системы.
* **Выбор инструментальных средств:** Был обоснован выбор инструментальных средств разработки, учитывая требования к производительности, совместимости, удобству отладки и наличию документации. Выбор операционной системы Windows, языка программирования F++ и Designer с встроенным компилятором языка обеспечивает доступ к обширной библиотеке объектов и мощным инструментам для разработки драйверов. Использование системы управления версиями Tortoise SVN обеспечивает эффективную совместную работу и отслеживание изменений в коде.
* **Проектирование пользовательского интерфейса:** Был спроектирован пользовательский интерфейс, ориентированный на простоту, понятность и функциональность. В частности, была реализована поддержка сетевого подключения к эмбоссеру по IP-адресу, что расширяет возможности интеграции драйвера в различные сетевые среды. Добавление соответствующих элементов управления (поля ввода IP-адреса и флажка выбора режима подключения) обеспечивает удобство настройки параметров подключения.

**Основные итоги главы:**

* Сформированы полные и детальные функциональные требования, необходимые для дальнейшей разработки драйвера.
* Разработана функциональная схема, обеспечивающая наглядное представление архитектуры и логики работы драйвера.
* Обоснован выбор оптимальных инструментальных средств разработки, обеспечивающих эффективность и надежность процесса разработки.
* Спроектирован пользовательский интерфейс, обеспечивающий удобство настройки и управления драйвером.
* Была реализована поддержка IP подключения эмбоссера, что значительно расширяет возможности интеграции.

Таким образом, данная глава заложила прочный фундамент для дальнейшей разработки драйвера прямого управления эмбоссером.

# 3 Описание и порядок работы с разработанным программным обеспечением

Эта глава посвящена подробному описанию разработанного драйвера прямого управления эмбоссером. В ней будут рассмотрены архитектура драйвера, его компоненты, интерфейс программирования (API), процесс установки и настройки, а также примеры использования для выполнения записи информации на карту. Целью этой главы является предоставление полного руководства для разработчиков и пользователей, желающих интегрировать и использовать разработанный драйвер в своих приложениях.

-- порядок работы общее описание --

## 3.1. Архитектура драйвера

Как было отмечено в прошлых главах, все драйверы для работы с кардридерами имеют одинаковую общую структуру, которой обязательно необходимо придерживаться, чтобы запускающий драйвер записи карты мог его подгрузить и использовать. В данном разделе сначала будет рассмотрена общая для всех таких драйверов структура и ее реализация в разработанном драйвере, а потом будут рассмотрены все дополнения к этой структуре, необходимые для работы драйвера.

Общая для драйверов работы с кардридерами структура объектов:

* Функция RInt Send(RStr sendData, LStr recvData) для отправки данных на подключенный ридер.
* Функция void Wait() для приостановки обработки и ожидания продолжения
* Функция void Open(RStr address) для осуществления подключения к ридеру
* Функция void Restart() для перезагрузки ридера и возврат его в изначальное состояние

Индивидуальные элементы разработанного драйвера, необходимые для его работы:

* CardConnect()
* CardClose()
* EjectCard()
* Port
* ReaderNum
* ReconnectSCR()
* Reset()
* sAddress
* TCPSend()
* WaitCard()

(Повторите кратко архитектуру драйвера, которая была подробно описана в предыдущей главе. Это необходимо для контекста. Пример:

Разработанный драйвер имеет модульную архитектуру, состоящую из следующих основных компонентов:

Менеджер соединений: Управляет установлением и завершением соединения с эмбоссером.

Обработчик команд: Преобразует команды API в специфичные для эмбоссера команды.

Драйвер устройства (Device Driver): Непосредственно взаимодействует с аппаратным интерфейсом.

Обработчик ошибок: Обрабатывает ошибки и уведомляет вызывающее приложение.

Менеджер конфигурации: Управляет настройками конфигурации драйвера.

Каждый компонент выполняет определенную функцию и взаимодействует с другими компонентами через четко определенные интерфейсы.

## 3.2. Описание API (Интерфейса программирования)

В этом разделе подробно описывается API драйвера, предоставляющий набор функций для управления эмбоссером.

### 3.2.1. Обзор классов и функций API

(Перечислите основные классы и функции API с кратким описанием их назначения и параметров.)

EmbosserDriver (Класс):

EmbosserDriver(): Конструктор класса.

~EmbosserDriver(): Деструктор класса.

Connect(InterfaceType interfaceType, Configuration configuration): Устанавливает соединение с эмбоссером.

Параметры:

InterfaceType: Тип интерфейса (USB, Serial, Ethernet).

Configuration: Объект, содержащий параметры конфигурации для соединения (например, COM-порт, IP-адрес).

Возвращает: Код успеха или код ошибки.

Disconnect(): Завершает соединение с эмбоссером.

Возвращает: Код успеха или код ошибки.

LoadCard(): Загружает карту в эмбоссер.

Возвращает: Код успеха или код ошибки.

EjectCard(): Выгружает карту из эмбоссера.

Возвращает: Код успеха или код ошибки.

EmbossCharacter(char character, Position position): Эмбоссирует символ в указанной позиции.

Параметры:

character: Символ для эмбоссирования.

position: Объект, определяющий позицию символа на карте (x, y).

Возвращает: Код успеха или код ошибки.

GetStatus(EmbosserStatus& status): Получает статус эмбоссера.

Параметры:

status: Ссылка на объект EmbosserStatus, в который будет записан статус.

Возвращает: Код успеха или код ошибки.

Configure(Parameter parameter, Value value): Устанавливает параметр конфигурации драйвера.

Параметры:

parameter: Параметр для настройки.

value: Значение параметра.

Возвращает: Код успеха или код ошибки.

SetErrorHandler(ErrorHandlerFunction errorHandler): Устанавливает функцию обратного вызова для обработки ошибок.

Параметры:

errorHandler: Указатель на функцию обработчика ошибок.

EmbosserStatus (Структура/Класс):

IsReady: Булевское значение, указывающее, готов ли эмбоссер к работе.

HasCard: Булевское значение, указывающее, есть ли карта в эмбоссере.

RibbonStatus: Статус ленты (если применимо).

ErrorCode: Код ошибки.

ErrorDescription: Описание ошибки.

(Другие классы и функции, специфичные для вашего драйвера)

## 3.2.2. Подробное описание наиболее важных функций

(Дайте более подробное описание 2-3 наиболее важных функций, включая примеры кода.)

Connect(InterfaceType interfaceType, Configuration configuration):

Описание: Устанавливает соединение с эмбоссером.

Параметры:

InterfaceType interfaceType: Тип интерфейса (USB, Serial, Ethernet). Определяется перечислением: enum InterfaceType { USB, Serial, Ethernet }.

Configuration configuration: Объект, содержащий параметры конфигурации для соединения. Тип Configuration зависит от типа интерфейса. Например, для Serial: struct SerialConfiguration { string ComPort; int BaudRate; }.

Возвращаемое значение: Возвращает DRV\_SUCCESS (0) при успешном соединении, или код ошибки при неудаче. Коды ошибок определены в enum ErrorCode.

Пример использования (C++):

#include "EmbosserDriver.h"

int main() {

EmbosserDriver driver;

SerialConfiguration serialConfig;

serialConfig.ComPort = "COM1";

serialConfig.BaudRate = 9600;

ErrorCode result = driver.Connect(InterfaceType::Serial, serialConfig);

if (result == DRV\_SUCCESS) {

std::cout << "Соединение установлено успешно!" << std::endl;

} else {

std::cerr << "Ошибка соединения: " << result << std::endl;

}

return 0;

}

c++

(Пример кода для EmbossCharacter или другой ключевой функции)

## 3.3. Процесс установки и настройки драйвера

В этом разделе описывается процесс установки и настройки разработанного драйвера в операционной системе.

### 3.3.1. Системные требования:

Операционная система: (Укажите поддерживаемые операционные системы)

Аппаратное обеспечение: (Укажите минимальные требования к аппаратному обеспечению)

Зависимости: (Укажите необходимые библиотеки или компоненты, которые должны быть установлены)

### 3.3.2. Порядок установки:

(Опишите шаги по установке драйвера. Например:

Запустите установочный файл setup.exe.

Следуйте инструкциям на экране.

Выберите каталог установки.

Подключите эмбоссер к компьютеру.

Операционная система должна автоматически обнаружить новое устройство и установить драйвер.

Если драйвер не устанавливается автоматически, укажите путь к каталогу с драйвером вручную через диспетчер устройств.

)

### 3.3.3. Настройка драйвера:

(Опишите способы настройки драйвера. Например:

Настройка параметров соединения: Укажите COM-порт, скорость передачи данных или IP-адрес эмбоссера через графический интерфейс настройки (если таковой имеется), или через конфигурационный файл.

Настройка параметров инициализации: При необходимости, укажите параметры инициализации эмбоссера, специфичные для конкретной модели.

)

## 3.4. Примеры использования драйвера

В этом разделе приводятся примеры кода на языке *(Укажите язык, например, C++)*, демонстрирующие использование драйвера для выполнения основных операций с эмбоссером.

### 3.4.1. Установление соединения и получение статуса:

#include "EmbosserDriver.h"

#include <iostream>

int main() {

EmbosserDriver driver;

SerialConfiguration serialConfig;

serialConfig.ComPort = "COM1";

serialConfig.BaudRate = 9600;

ErrorCode result = driver.Connect(InterfaceType::Serial, serialConfig);

if (result != DRV\_SUCCESS) {

std::cerr << "Ошибка соединения: " << result << std::endl;

return 1;

}

EmbosserStatus status;

result = driver.GetStatus(status);

if (result != DRV\_SUCCESS) {

std::cerr << "Ошибка получения статуса: " << result << std::endl;

driver.Disconnect();

return 1;

}

std::cout << "Статус эмбоссера:" << std::endl;

std::cout << " Готов: " << (status.IsReady ? "Да" : "Нет") << std::endl;

std::cout << " Карта в устройстве: " << (status.HasCard ? "Да" : "Нет") << std::endl;

driver.Disconnect();

return 0;

}

c++

## 3.4.2. Загрузка карты и эмбоссирование символа:

// ... (Предыдущий код для соединения) ...

result = driver.LoadCard();

if (result != DRV\_SUCCESS) {

std::cerr << "Ошибка загрузки карты: " << result << std::endl;

driver.Disconnect();

return 1;

}

Position position;

position.X = 10;

position.Y = 20;

result = driver.EmbossCharacter('A', position);

if (result != DRV\_SUCCESS) {

std::cerr << "Ошибка эмбоссирования: " << result << std::endl;

driver.EjectCard();

driver.Disconnect();

return 1;

}

// ... (Код для выгрузки карты и завершения соединения) ...

c++

(Добавьте другие примеры использования для ключевых функций)

## 3.5. Обработка ошибок

(Опишите, как драйвер сообщает об ошибках, какие коды ошибок используются, и как вызывающее приложение должно обрабатывать ошибки.)

3.5.1. Коды ошибок:

(Приведите таблицу с кодами ошибок и их описанием)

3.5.2. Функция обратного вызова (Callback):

(Опишите, как можно использовать функцию обратного вызова для асинхронной обработки ошибок)

## 3.6. Заключение

В этой главе было в своих приложениях. В следующей главе будут представлены результаты тестирования драйвера и оценка его производительности.

## 3.1 Качественный анализ разработанного драйвера в тестовой среде

Никакой качественный анализ не обходится без тестирования разработанного программного продукта в реальной среде. Настроив новый карточный продукт под запись карты «Введите название карты и платежную систему» в TranzWare CardFactory и создав новый пакет для обработки карты, в интерфейсе пакета возможно запустить обработку карточного продукта. После запуска и успешной отработки всех модулей логи обработки будут заполнены и весь дальнейший анализ будет основываться опираясь на них.

В логах обработки нас интересует только раздел, посвященный логам модуля ChipWriterDriver, который и осуществлял в данном случае работу с разработанным драйвером. Этот раздел начинается с лога «Введите текст лога», поэтому найти его не составит труда. Внутри данного раздела можно увидеть когда и какие сообщения передавались карте и принимались от карты.

## Передача драйвера клиентам для тестирования

После того, как драйвер был успешно разработан и протестирован внутри компании, необходимо будет внести данный драйвер в ближайшую новую версию приложения для передачи клиентам. После внесения и тестирования приложения в клиентской среде, клиентами будут переданы логи обработки для дальнейшего анализа. После проведения финального анализа продукта, прошедшего проверку в клиентской среде, можно будет считать, что продукт успешно разработан и выпущен в работу и теперь войдет в фазу поддержки

### 2.3.6. Дополнительные инструменты

Кроме всего вышеперечисленного, при разработке драй

## 2.1 Анализ имеющихся средств разработки драйверов

Как уже упоминалось в первой главе, программная платформа FloraWare предоставляет большие возможности для разработки любого ПО, в том числе связанного с сетевым и межаппаратным взаимодействием. Сам процесс разработки и доступ ко всем возможностям программной платформы происходит через встроенное приложение “Designer”, в будущем – Дизайнер. Оно позволяет с нуля создать любой программный продукт и запустить его в работу в среде FloraWare.

Создание программного продукта в среде Дизайнера начинается с создание нового файла-хранилища с расширением «.ots». Средствами Дизайнера разработчик будет создавать объекты, которые потом будут записываться в файл и подгружаться по мере надобности. Объекты в среде FloraWare необходимы для осуществления различных целей, таких как создание массивов констант, выполнения программного кода, запросов к базе данных, отображения различных графических элементов и т.д. Практически любой элемент программы может быть создан при помощи объектов, будь то структура данных, программная функция или же элемент графического интерфейса. Весь процесс разработки состоит из построения дерева данных объектов и создания связей между ними, к примеру создав элемент кнопки, возможно привязать к событию нажатия на нее выполнение программного кода, а также одной или нескольких объектов-функций, созданных в дереве, или же наоборот, внутри объекта-функции будет происходить взаимодействие с объектом-кнопкой в дереве, в виде изменения графической составляющей кнопки, отключения ее функционала или же внесения каких-либо других изменений.

При создании программного продукта Дизайнер предлагает удобный интерфейс, множество встроенных средств тестирования, а также большой набор встроенных объектов. Все объекты внутри представленной среды разработки можно разделить на несколько категорий:

1. графические элементы,
2. структуры для хранения данных,
3. элементы, взаимодействующие с базами данных,

Каждая из этих категорий предлагает целый набор элементов для осуществления специализированной для категории деятельности. По мере надобности в рамках данной работы будут упоминаться объекты, относящиеся к представленным категориям.

## 2.2 Разработка драйвера взаимодействия с эмбоссером

Разработка драйвера взаимодействия с эмбоссером будет проходить в два этапа – проектирование и разработка в Дизайнере, за которыми будут следовать подключение драйвера в CardFactory и тестирование как внутри компании, так и на стороне клиентов. На этапах проектирования и разработки будет учитываться структура уже существующих драйверов взаимодействия с эмбоссерами, а также информация, представленная в документации компании «MATICA», описанная в первой главе.

Проектирование структуры драйвера предполагает понимание роли драйвера в схеме взаимодействия с эмбоссером. Драйвер по большому счету будет представлять из себя только модуль передачи команд эмбоссеру от приложения, то есть на вход драйвер будет получать передаваемое карте сообщение, после будет вносить это сообщение в подготовленную строку кода на языке TCPScript и отправлять ее на эмбоссер по сетевому протоколу. Выходом же драйвера будет полученное в результате выполнения операции сообщение от карты в виде двух байт, которые определяют успешность выполнения операции или же наличие возникших в результате выполнения ошибок.

Анализируя структуру схожих драйверов, можно определить, что внутри драйвера обязательно необходимо иметь два объекта-функции send и receive, которые будут осуществлять отправку данных эмбоссеру и принятие ответа от него. Эти функции будут использоваться внешним модулем, который будет подгружать драйвер и использовать их при выборе эмбоссера MATICA как основного эмбоссера для записи.

## 2.3 Подключение драйвера в приложение

После того, как драйвер был разработан необходимо подключить его к инфраструктуре CardFactory, для того, чтобы входящие в него модули могли подключать его и использовать входящий в него функционал. Для таких целей существует отдельная утилита, с помощью которой можно управлять проектами, созданными на основе платформы FloraWare – Project Manager. Открыв файл проекта CardFactory (cardfactory.prj) через Project Manager можно увидеть всю древовидную структуру модулей приложения. Каждое приложение имеет связи с различными библиотеками, которые созданы в качестве модулей. Чтобы другие модули могли обнаружить модуль драйвера и подключить его в свою экосистему нужно добавить его в нужную ветку проекта, выполнить правки в функции подключения модуля управления эмбоссером, а также внести изменения в интерфейс модуля выбора эмбоссера для включения возможности укзания базового IP-адреса машины, к которой будет подключатся приложение.

# Заключение

# Список используемой литературы

# Приложения