Техническая Архитектура Программного Обеспечения для работы с программируемыми логическими контроллерами (ПЛК)

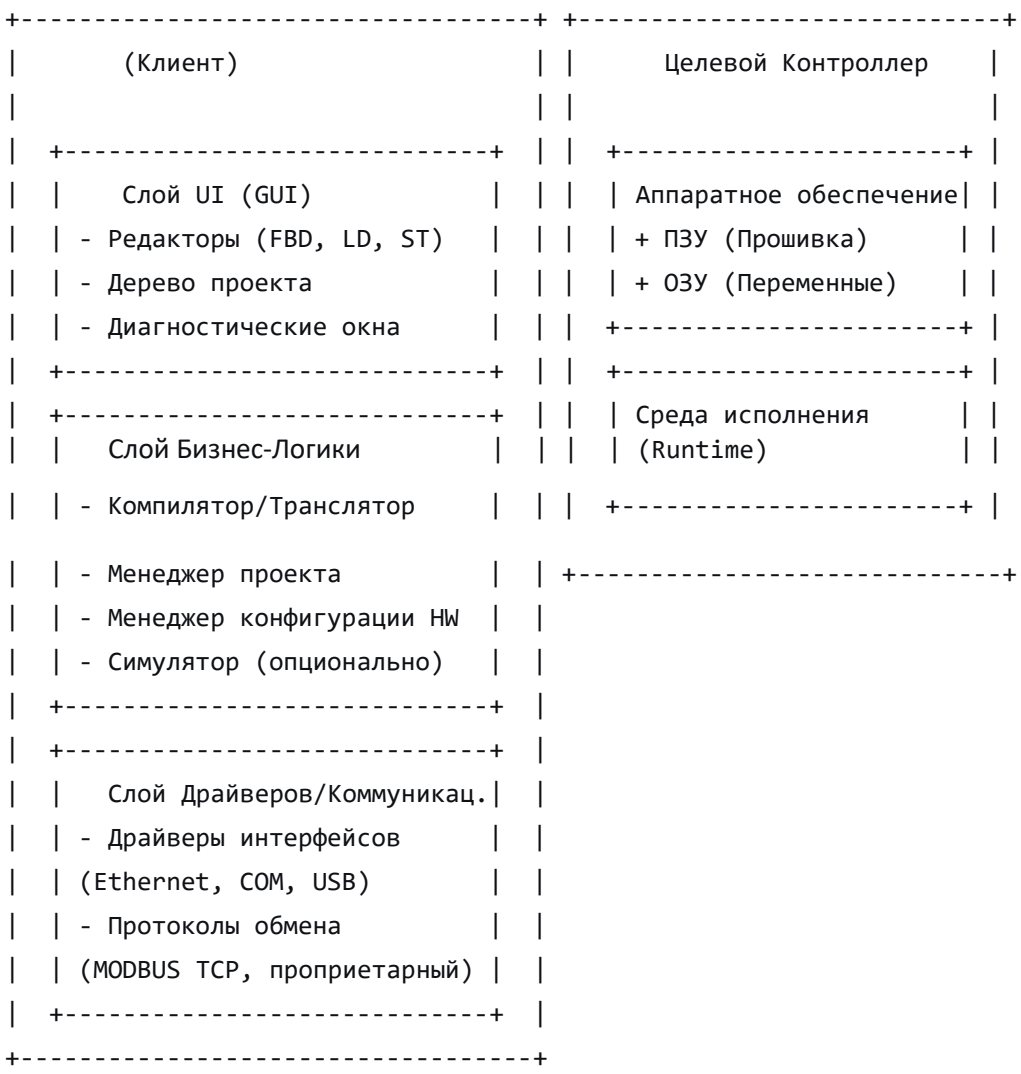
1. Введение и Назначение

Основная задача — предоставить инженеру единый инструмент для конфигурирования аппаратных средств, программирования управляющих алгоритмов, отладки и диагностики оборудования.

2. Высокоуровневая Архитектура (Многоуровневая Архитектура)

ПО построено по принципу **многоуровневой (слоеной) архитектуры**, что обеспечивает модульность, простоту поддержки и дальнейшего развития. Архитектуру можно представить в виде следующих слоев:

- **Слой Пользовательского Интерфейса (UI Layer)**
- **Слой Бизнес-Логики (Business Logic Layer)**
- **Слой Драйверов и Коммуникации (Driver/Communication Layer)**
- **Слой Внешних Контроллеров (Controller Layer)**



3. Детальное Описание Компонентов Архитектуры

3.1. Слой Пользовательского Интерфейса (UI Layer)

- **Технология:** используется кроссплатформенный фреймворк, такой как **Qt (C++/QML)** или **.NET (WPF/C#)**, что позволяет работать под ОС Windows и Linux.
- **Компоненты:**
    - **Главное окно с панелями инструментов:** Стандартный интерфейс MDI (Multiple Document Interface) или с плавающими панелями.
    - **Редакторы языков МЭК 61131-3:**
        - **FBD (Function Block Diagram):** Визуальный редактор для построения схем из функциональных блоков.
        - **LD (Ladder Diagram):** Редактор релейно-контактных схем (лестничных диаграмм).
        - **ST (Structured Text):** Текстовый редактор с подсветкой синтаксиса и авто-дополнением.

      При реализации программы RU-DRIVE GT-control использован язык – ST (Structured Text) из перечня МЭК 61131-3 промышленных языков программирования ПЛК (программируемых логических контроллеров). Весь исходный код программного обеспечения RU-DRIVE GT-control написан на этом языке.

    - **Редактор конфигурации аппаратуры (Hardware Configurator):** Графический инструмент для "сборки" стойки контроллера: добавления CPU, модулей ввода-вывода (AI, AO, DI, DO), коммуникационных модулей и настройки их параметров (адреса, диапазоны, фильтры).
    - **Дерево проекта (Project Tree):** Иерархическое отображение всех компонентов проекта: устройств, программ, задач, глобальных переменных.
    - **Окно диагностики и мониторинга (Watch Window):** Таблицы для онлайн-просмотра и изменения значений переменных контроллера.
    - **Окно сообщений (Output Window):** Вывод логов компиляции, ошибок, статуса соединения.

3.2. Слой Бизнес-Логики (Business Logic Layer)

Это "мозг" приложения, где происходит вся обработка данных.

- **Компоненты:**
    - **Компилятор/Транслятор (Compiler):**
        - Преобразует визуальные и текстовые программы (FBD, LD, ST) в промежуточный код (например, на языке Ассемблера

контроллера) или в специальный байт-код, понятный среде исполнения контроллера.
        - Проводит синтаксический и семантический анализ кода.
        - Создает таблицы символов (переменных) и связывает их с физическими адресами ввода-вывода.
    - **Менеджер проекта (Project Manager):**
        - Управляет структурой проекта, файлами конфигурации, ресурсами.
        - Отвечает за сериализацию (сохранение) проекта в файлы на диск и десериализацию (загрузку).
    - **Менеджер конфигурации аппаратуры (HW Config Manager):**
        - Верифицирует корректность созданной конфигурации (например, проверка совместимости модулей, занятости слотов, конфликтов адресов).
        - Генерирует бинарный файл конфигурации, который загружается в ПЗУ контроллера.
    - **Диспетчер связи (Communication Manager):**
        - Координирует работу между UI и драйверами связи. Формирует запросы для чтения/записи данных.
    - **Симулятор (Simulator) :**
        - Программная эмуляция работы контроллера и его модулей ввода-вывода. Позволяет отлаживать логику программы без наличия физического "железа".

3.3. Слой Драйверов и Коммуникации (Driver/Communication Layer)

Этот слой отвечает за физическое и логическое взаимодействие с контроллером.

- **Технологии:**
    - **Драйверы интерфейсов:** Стандартные системные драйверы для Ethernet, последовательных портов (COM/USB).
    - **Сетевые сокеты (Sockets):** Для работы по Ethernet.
- **Протоколы обмена данными:**
    - Скорее всего, используется **проприетарный протокол поверх TCP/IP** (для моделей с Ethernet-портом), оптимизированный для надежной и быстрой передачи конфигурации, программ и циклического обмена данными.
    - Для более старых или простых моделей может использоваться **MODBUS TCP** или **MODBUS RTU** (для COM-порта) как стандартный и универсальный протокол.
    - Этот слой инкапсулирует знание о том, как "упаковать" команду компилятора или запрос на чтение переменной в кадр передаваемых данных.

3.4. Слой Внешних Контроллеров (Controller Layer)

- **Компоненты на контроллере:**
    - **Прошивка (Firmware):** Постоянное ПО, "прошитое" в ПЗУ контроллера. Включает в себя:
        - **Загрузчик (Bootloader):** Принимает новую конфигурацию и программу от ПО и записывает их в память.
        - **Среда исполнения (Runtime):** Интерпретирует или выполняет скомпилированный байт-код, ПО. Управляет циклом выполнения задач, опросом модулей ввода-вывода, обработкой прерываний.
        - **Стек коммуникационных протоколов:** Обеспечивает прием и передачу данных по тому же протоколу, что и ПО.

4. Ключевые Технологии и Принципы

- **Кроссплатформенность:** Современные требования диктуют необходимость работы не только под Windows.
- **Модульность:** Каждый редактор, компилятор, драйвер — это отдельный модуль (плагин), что упрощает разработку и обновление.
- **Расширяемость:** Архитектура должна позволять легко добавлять поддержку новых модулей ввода-вывода или даже новых семейств контроллеров.
- **Надежность:** Использование транзакционных механизмов при загрузке проекта в контроллер (например, проверка контрольных сумм, откат при ошибке).
- **Безопасность:** Возможность реализации механизмов защиты проекта паролем, разграничения прав пользователей.

5. Процесс Работы (на примере загрузки проекта)

1. **Разработка:** Инженер создает конфигурацию аппаратуры и программу на ST в UI-слое.
2. **Компиляция:** Слой бизнес-логики компилирует проект. При успехе генерируется бинарный файл прошивки и файл отладочной информации (символов).
3. **Установление связи:** ПО через слой коммуникации устанавливает соединение с контроллером (например, по IP-адресу).
4. **Загрузка:** Бинарный файл передается через драйвер в загрузчик контроллера. Контроллер перезапускается и начинает выполнять новую программу под управлением своей среды исполнения.
5. **Мониторинг:** Инженер использует окно диагностики в UI. Запросы на чтение переменных отправляются через слой коммуникации, получают ответ от Runtime контроллера, и данные отображаются в интерфейсе.

Данная архитектура является классической для ПО данного класса и обеспечивает необходимую надежность, производительность и легкость в поддержке и развитии.