Описание СПО Пересмешник

Введение.

В подавляющем большинстве проектов, ПО должно взаимодействовать с большим количеством приборов и устройств. Каждое устройство имеет свой интерфейс подключения для обмена информацией, часто даже не один. Как правило, каждое устройство имеет свой собственный формат сообщений и протокол обмена, основанный на нём. Ещё одной распространённой проблемой является физическое отсутствие устройства в период проектирования и разработки.

Обоснование.

Исходя из сказанного, предлагается разработать ПО, которое должно облегчить создание программных модулей для работы с устройствами, а так же позволит вести её даже в случае физического отсутствия устройств.

Предполагается, что ПО будет состоять из библиотеки классов для поддержки необходимых концепций/абстракций (Транспорт, Формат, Сообщение/Команда, Протокол), системы описания протоколов и программы для имитации приборов и устройств на основе этих описаний. Предполагается, что описания будут создаваться либо с помощью пользовательского интерфейса, либо вручную. Результатом будут файлы базовых классов и программа, настроенная на имитацию работы устройств. Ещё одним

плюсом видится, что источником информации будут одни и те же настроечные файлы и для автоматического создания кода и для экземпляра программы имитирующей заданное устройство/прибор.

Библиотека классов.

Библиотека должна состоять из классов, реализующих на высоком уровне понятия Транспорт, Формат сообщения/команды, Сообщение/Команда, Протокол, а так же интерпретирующих их описания созданных с помощью системы описания информационного взаимодействия.

Примечание: При разработке рекомендуется использовать стандартные классы из библиотек Qt. Например, при реализации понятия Транспорт следует рассмотреть использования классов на основе QIODevice

.

Система описания протоколов.

Предполагается, что система описания протоколов должна состоять из следующих частей:

Транспорт.

Модуль предназначен для настройки параметров транспорта, по которому предполагается работа протокола. Под транспортом понимаются различные способы обмена/получения информации (через серийные порты, TCP/UDP соединения – как клиенты, так и сервера). Транспорт как способ передачи информации предполагается сделать полностью независимым от формата сообщений и протокола обмена.

Формат команд/сообщений.

Сообщения протоколов могут иметь различный формат, который надо постараться формализовать.

Сообщения – из нашей практики - бывают:

- фиксированной длины.
- переменной длины с маркерами начала/конца сообщения, как, например, в сообщениях 14Ц821 (“Грот-В”).
- произвольные последовательности (двоичные или текстовые)
- переменной длины, где длина задаётся полем в самом сообщении или предваряющем его.
- условно переменной длинны, где длина сообщения (как правило от устройства) может быть разной в зависимости от типа возвращаемого значения.

Практически все виды форматов сообщений включают в себя уникальный код (например, часто это 0xAA) и код команды (как правило, в начале), а так же контрольной суммы (как правило, в конце командной последовательности). В некоторых используется экранирование/преобразование спецсимволов (как правило, маркеров начала/конца сообщения), как, например, в сообщениях для 14Ц821 (“Грот-В”). В некоторых случаях может присутствовать контрольная сумма для заголовка

Команды протокола работы с устройством.

На основе команд определенного формата описанного тем или иным образом (см. выше) формируются команды/сообщения для конкретного протокола.

Сообщения (команды) бывают нескольких типов:

- Команды для настройки устройства, с последующим получением ответного сообщения от устройства с кодом завершения (например, успешно или нет).
- Команда, инициирующая приём данных. Может тоже возвращать сообщение о готовности устройства передать данные, либо нет.
- Сообщение с данными (как правило, от устройства).

Логика работы устройства

(необходимый минимум достаточный для тестирования) на основе команд и их последовательностей в соответствии с протоколом. Имея в своём распоряжении описания команд протокола можно попытаться описать логику работы устройства для некоторых случаев. Например, мы можем послать сообщения установить определённые параметры устройства и ожидать ответного сообщения об успехе. Или может быть послан запрос о начале приёма данных от устройства.

Примечание: Логику работы устройства, возможно, следует реализовать как конечный автомат с использованием классов Qt из подмножества State Machine Framework

Для каждой части системы описания предлагается иметь свой специализированный (проблемно-ориентированный) язык (предлагается использовать XML, но в случае описания для транспорта, например, будет достаточно что-то вроде файла настроек Qt). Для каждого такого языка нужно будет реализовать разборщик и классы поддерживающие необходимую функциональность.

Примеры предметно-ориентированных языков (ПОЯ).

Попробуем привести примеры возможных реализаций предметно-ориентированных языков.

Настройки транспорта.
Необходимы для программы-имитатора информационного обмена и программы-клиента. Пример как это может выглядеть для случая TCP (простейший из возможных вариантов транспорта):

```
[Server]
IP=127.0.0.1
Type=TCP
Port=1234
```

Формат сообщения.

Например, формат для команд/сообщений фиксированной длинны может быть описан следующим образом:

```
<OESFormat>
        <Transform name="OES_ByteStuffing">
                <replace from="0xAA" to="0xBB 0x00"/>
                <replace from="0xBB" to="0xBB 0x01"/>
        <Transform/>
        <Format name="OES_Screen_Activator" transform="OES_ByteStuffing">
                <MSG name="CMD">
                        <ID>0xAA</ID>
                        <COM/>
                        <ADR size="2">5</ADR>
                <DAT size="2"/>
                        <RESER/>
                        <CRC256/>
                </MSG>
                <MSG name="DATA">
                        <DATA type="byteArray">
                        <CRC256/>
                </MSG>
        </Format>
</OESFormat>
```

Примечание: Поля с заданными значениями могут использовать эти значения для заполнения по умолчанию. При этом CRC256 в данном примере (контрольная сумма по модулю 256) является одним из зарезервированных слов описания Команд/Сообщений протокола.

Следует заметить, приведенное выше описание может считаться сообщением фиксированной длинны лишь условно, так как используется замена специальных значений на условные последовательности при отсылке сообщений. Обратная операция производится при получении ответа от устройства.

Например, в другой системе (назовём её Lab) для команды ожидания готовности спектральной информации может быть использовано следующее описание:

```
<REQ name="GET_DARK" format="OES_Lab">
    <COM>0x03</COM>
        <RET name="Code">
        <DAT byte="Lo">
        </RET>
</REQ>
```

Обратите внимание, что ответное сообщение описано следующим образом:

```
<RET name="Code">
    <DAT byte="Lo">
 </RET>
```

Предполагается, что в результате в созданном классе будет сформирован метод GetCode(), возвращающий значение младшего байта (тип в C++ «unsigned char») поля DAT (5-ый с начала или D_L стандартной команды в – условно - 8-ми байтовом формате ОЭС).

Теперь опишем целиком небольшую систему команд для платы активации

```
<Activator>
 <REQ name="IS_DISFUNCTIONAL" format="OES_Activator">
    <COM>0x01</COM>
    <RET>
        <DAT byte="Lo" >
        <option value="0">FUNCTIONAL</option>
        <option value="1">DISFUNCTIONAL</option>
        </DAT>
    </RET>
</REQ>
<REQ name="START_ACTIVATION" type="CMD" format="OES_Activator">
    <COM>0x02</COM>
    <DAT name="activationDelay"/>
 </REQ>
<REQ name="GET_SERIAL_NUMBER" type="CMD" format="OES_Activator">
    <COM>0x03</COM>
    <RET type="ByteArray">
        <ADR byte="Hi"/>
        <DAT byte="Lo"/>
        <DAT byte="Hi"/>
        <RESER/>
    </RET>
</REQ>
</Activator>
```

В результате после разбора этих описаний от генератора кода мы должны будем получить набор специализированных классов.

Описания логики работы устройства.

Если будет решено использовать конечный автомат в качестве основной идеи для реализации модуля, то возможно следует рассмотреть использование State Chart XML (SCXML), хотя желательно попытаться применить какие-то более простые формы описания.

Программа имитации информационного обмена с устройством.

Программа имитации информационного взаимодействия для каждого конкретного устройства будет получать на вход файлы описания для всех перечисленных выше модулей (путем задания параметров из командной строки и/или через файлы конфигурации). Эти файлы и будут определять поведение программы.

Автоматическое создание структур данных и служебных классов (генератор кода).

Для облегчения работы разработчиков, а так же улучшения качества и скорости разработки, следует рассмотреть возможность автоматического создания файлов структур данных (структур, перечислений и констант различных видов) и служебных классов на основе имеющихся файлов описаний всех уровней. Необходимо продумать возможность такого их создания, чтобы внесение изменений минимально влияло на процесс разработки (заголовочные файлы, абстрактные классы, шаблоны и т.п.).

Пользовательский интерфейс разработки.

Для создания описаний каждой подсистемы предполагается использовать оконный интерфейс. Соответственно – в идеале – каждый модуль должен иметь пользовательский интерфейс, состоящий из одного или нескольких окон.

Транспорт.

- Диалоговое окно настройки конкретного транспорта (номер и параметры обмена серийного порта, номер порта и адрес сервера для протоколов TCP/IP, параметры USB порта).

Окно описания формата посылок.

- Диалоговое окно, позволяющее задать описание особенностей формата протокола.

Протокол.

- Диалоговое окно, описывающее команды/сообщения протокола.

Настройка программы-имитатора устройства.

Для настройки экземпляра программы-имитатора конкретного устройства необходимо настроить его транспорт (диалоговое окно Транспорт), формат команд/сообщений (диалоговое окно Описания формата) и команды/сообщения протокола (окно Протокол) и их конкретное применения. Например, мы должны иметь возможность задать пары команд/сообщений типа «установка параметров» - «код выполнения» (Успех/Неуспех/Код ошибки), где первая из команд посылается из приложения в программу-имитатор

устройства, соответственно ожидая один из вариантов второго сообщения от программы-имитатора. После настройки параметров экземпляр может быть запущен из командной строки с необходимыми параметрами или из окна запуска с перечнем настроенных протоколов для устройств пользовательского интерфейса. Так как транспорт предполагается сделать независимым (он всего лишь передаёт информацию в определённом формате от программы-клиента устройства программе-имитатору устройства и обратно) от остальных модулей, то на начальном этапе мы можем реализовать самый простой вариант (например, в виде локального сервера TCP/IP). Кроме этого, интерфейс должен будет выступать в роли «помощника» для создания базовых классов согласно описаниям для подсистем, которые могут быть использованы в программе как основа для быстрой разработки.

Примерное разбиение на подзадачи (Начальный этап на примере протокола ОЭР в сроки, отведённые существующим планом).

Примечание: Задачи перечисляются по предлагаемому порядку их исполнения внутри проектного модуля.

Библиотека классов.

Базовые классы и шаблоны для создания пользовательских классов.
Реализация обмена данными клиент-сервер по транспорту TCP/IP (TCP).
Служебные классы для классов Формат, Команды/Сообщения.
Прототип программы имитации информационного взаимодействия.

Предметно-ориентированные языки (ПОЯ).

Необходимо разработать следующие ПОЯ и классы для их обработки:

В первую очередь
- Команд/Сообщений протокола.
- Формата сообщения.

Потом
- Описания логики работы устройства.

И в последнюю очередь (как самый простой и легко воспроизводимый в программе):
- Настройки транспорта устройства для программы-имитатора и программы-клиента.

Программа имитации информационного взаимодействия (прототип).

Необходимо разработать консольное приложение с выводом отладочной информации о подключении клиентов и о взаимодействии с ними (получение и отправка сообщений). В дальнейшем нужно рассмотреть возможность написания оконного приложения или сервиса (демона) с возможностью подключения через консоль/окно.