УДК 004.415.2.031.4 3

Рыбалёв Андрей Николаевич
Амурский государственный университет
г. Благовещенск, Россия
*E-mail:*amgu_appe@mail.ru
Rybalev Andrey Nikolaevich
Amur State University
Blagoveshchensk, Russia
*E-mail:*amgu_appe@mail.ru

ПРИМЕНЕНИЕ ТЕХНОЛОГИИ ООП ПРИ РАЗРАБОТКЕ ПРОГРАММ УПРАВЛЕНИЯ КОММУТАЦИОННЫМИ АППАРАТАМИ

OOP TECHNOLOGY APPLICATION IN THE DEVELOPMENT OF SWITCH DEVICES CONTROL PROGRAMS

Аннотация. Предложен подход к программной реализации системы управления коммутационными аппаратами на основе технологии объектно-ориентированного программирования.

Abstract. An approach to the software implementation of the control system for switching devices based on the technology of object-oriented programming is proposed.

Ключевые слова: коммутационный аппарат, программируемый логический контроллер (ПЛК), функциональный блок, интерфейс, наследование.

Key words: switching device, programmable logic controller, functional block, interface, inheritance.

DOI: 10.22250/20730268_2023_103_49

Введение

На многих промышленных предприятиях управление технологическим процессом осуществляется посредством многочисленных аппаратов, выполняющих те или иные коммутации потоков среды или энергии. Типичным примером является электрическая подстанция, на распределительных устройствах которой производятся различные переключения линий, шин, силовых и измерительных трансформаторов, а также другого оборудования. Основные виды коммутационных аппаратов – высоковольтные выключатели, разъединители и заземляющие ножи. Эти аппараты оснащены собственными приводами, которые допускают как местное, так и дистанционное управление. В последнем случае команды управления в виде дискретных сигналов поступают от специализированного контроллера, соединенного посредством тех или иных каналов связи с оборудованием диспетчерского пункта. В процессе управления контроллер получает информацию о состоянии аппарата и режиме его функционирования. Эта информация, также имеющая дискретный характер, передается диспетчеру.

Важным обстоятельством является то, что по условиям безопасности аппараты требуют согласованного управления. В частности, запрещается проводить операции с разъединителем под напряжением (при включенном выключателе), опускать заземляющий нож при включенном разъединителе и т.д. Соответствующие блокировки реализованы как на уровне электрических схем управления приводами аппаратов, так и в программе управляющего контроллера. Таким образом, ошибочные действия диспетчера, манипулирующего элементами управления на своем экране визуализации, не могут привести к фатальным последствиям.

Однако остается актуальной проблема формирования и автоматического выполнения так называемых *программ* или *планов переключений*, задающих последовательность срабатывания аппаратов определенной группы для решения какой-либо более крупной задачи, – например, отключения одной из линий электроснабжения и включения другой (резервной). В данной работе предложен подход к решению этой проблемы, основанный на использовании технологии объектно-ориентированного программирования, которая все более активно применяется в промышленной автоматизации [1].

Идея подхода

Для «классических» программ управления внешний мир (объект управления) по существу представлен совокупностью входов и выходов ПЛК. План переключений аппаратов в таком случае может быть представлен последовательными списками выходов, которые нужно активировать, и входов, которые нужно при этом контролировать. Такой план может быть составлен и понят только программистом и совершенно не годится для оперативного персонала, привыкшего к работе с сущностями более высокого уровня, т.е. с самими аппаратами.

На «пользовательском» уровне план может быть составлен в виде массива структур «аппарат – действие», а потом оттранслирован в списки входов-выходов ПЛК. Такой подход имеет право на жизнь, однако процедура трансляции представляется крайне трудоемкой, особенно при большом числе аппаратов. Потребуется вести «таблицы соответствия» и привлекать достаточно сложные алгоритмы для работы с ними.

Объектно-ориентированный подход легко решает эту проблему. Все аппараты можно представить отдельными программными *объектами* – экземплярами ограниченного числа *классов*, тогда входы-выходы ПЛК, привязанные к аппаратам, будут *данными* этих объектов. Никаких таблиц соответствия уже не потребуется, поскольку сами объекты обладают нужной информацией. *Методы* (в частности, реализующие управление аппаратом) определяются для класса, а не для конкретного объекта. Они не требуют знания специфической для объекта информации (например, привязки входов-выходов), точнее, они получат эту информацию в момент вызова от самого объекта, *для которого* они вызываются.

Однако план переключений в виде списка аппаратов не может быть представлен массивом объектов разных классов, поскольку массив может содержать только однотипные элементы. Здесь на помощь приходит концепции *наследования* и *полиморфизма*. Можно ввести некий суперкласс «Аппарат Как Таковой», который «содержит» некоторый базовый набор свойств и «умеет» выполнять некоторый базовый набор действий (а на самом деле ничего не содержит и ничего не умеет, – это просто декларация). В языке C++ такой класс называется *виртуальным базовым классом*. Все «реальные» аппараты, точнее, классы аппаратов, будут наследовать от класса «Аппарат Как Таковой», представляя собственные реализации свойств и методов. Тогда этим суперклассом можно «подменять» классы реальные. Можно создать массив указателей на суперкласс, фактически содержащий указатели на объекты реальных классов. При обработке этого массива будут вызываться методы реальных классов и обрабатываться данные реальных объектов.

Сущности

До последнего времени подход, изложенный выше, мог быть реализован только в программах для компьютеров, написанных на объектно-ориентированных языках типа C++ или Java. Фактически это означает, что составлением и, что более важно, выполнением плана переключений должна заниматься операторская станция, а не ПЛК. Представляется перспективным возложить решение этих задач на контроллер, оставив станции функции человеко-машинного интерфейса.

Одним из лидеров продвижения объектно-ориентированного подхода в практику программирования ПЛК является немецкая компания CODESYS GmbH (старое название 3S-Smart Software Solutions GmbH), входящая в группу компаний CODESYS Group [2]. Ее SoftLogic и runtime систему

CODESYS (Controller Development System) используют многие производители ПЛК по всему миру, в том числе российская компания ОВЕН [3] – ведущий отечественный разработчик средств промышленной автоматизации. В последних версиях системы в качестве расширения стандарта IEC 61131-3 [4] реализована поддержка практически всех основных элементов объектно-ориентированного программирования.

Виртуальные базовые классы в CODESYS 3.5 называются *интерфейсами*. Интерфейсы декларируют свойства и методы, которые должны реализовать наследующие им классы (функциональные блоки). Причем в отличие от C++ в CODESYS разделяются понятия «обычного» наследования (extending – расширение) и наследования от виртуального класса (implementation – реализация).

«Организационная структура» данных в первом приближении показана на рис. 1.

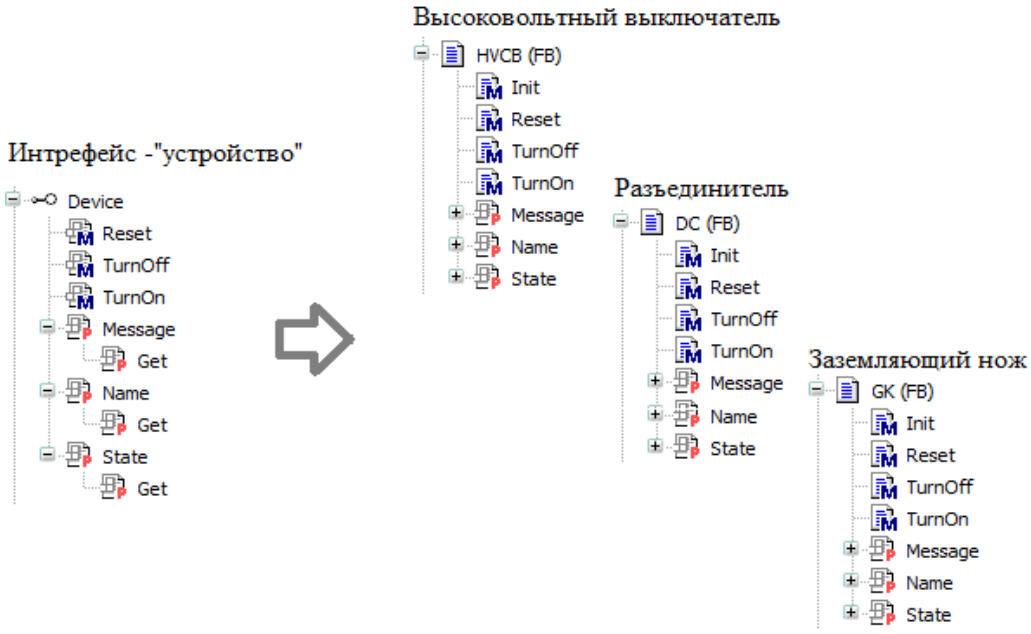

Рис. 1. Интерфейс и функциональные блоки.

Интерфейс Device («устройство»), реализуемый функциональными блоками HVCB (выключатель), DC (разъединитель), GK (заземляющий нож) и, возможно, другими, включает: метод Reset (сброс после ошибки); методы TurnOn и TurnOff (включить/выключить аппарат); свойство Message (последнее сообщение от аппарата); свойство Name (идентификатор аппарата); свойство State (состояние аппарата).

Для всех свойств реализации подлежат методы Get (прочитать значение), но не Set (установить).

В функциональные блоки добавлены методы Init, предназначенные для инициализации экземпляров. В CODESYS можно использовать и настоящие *конструкторы* – методы FB_Init, вызываемые при объявлении экземпляров, однако необходимость в наличии перекрестных ссылок мешает этому. Для решения своих задач экземпляры должны знать своих соседей, поэтому они сначала создаются (объявляются), а потом инициализируются.

Все экземпляры объявляются в глобальном списке и инициализируются специальной программой Initialization, которая вызывается однажды при старте системы.

Функциональные блоки содержат полный набор данных, необходимых для их работы. В качестве примера приведем список локальных переменных функционального блока DC.

```
FUNCTION_BLOCK DC IMPLEMENTS Device
VAR
        myName: STRING(10);
```

```
        myState: STRING(10):='off';
        myMessage: STRING(10);
        myHVCB: Device; //ссылка на выключатель
        myGK: Device;// ссылка на заземляющий нож
        //указатель на выход управления приводом – включить
    myActuator: POINTER TO BOOL;
    //указатель на выход управления приводом – выключить
    myDeactuator: POINTER TO BOOL;
    //указатель на вход – состояние аппарата(включен)
        myActivated: POINTER TO BOOL;
    //указатель на вход – состояние аппарата(выключен)
        myDeactivated: POINTER TO BOOL;
    //указатель на вход – режим управл. аппаратом (мест./дист.)
        myWorkMode: POINTER TO BOOL;
        timer:TON; //таймер для контроля времени операций
    END_VAR
```

Строковые переменные напрямую транслируются в соответствующие свойства их методами Get. Блок содержит ссылки на «соседей» – выключатель и заземляющий нож. Оба объекта представлены как интерфейсы Device, что позволяет коду блока читать их свойства и даже вызывать методы (последнее не используется). Связь с аппаратурой организуется посредством указателей на входы и выходы ПЛК.

Основную задачу – управление приводами и контроль состояния аппаратов – решает программная часть функциональных блоков. Она построена по автоматному принципу: в зависимости от состояния, в котором находится аппарат («включен», «выключен», «включается», «выключается», «ошибка» и др.), выполняются те или иные действия, сопровождаемые выдачей в свойство Message соответствующих сообщений. При этом, помимо собственного, контролируется также и состояние аппаратов-соседей. При достижении определенных условий производится смена состояния аппарата.

В отличие от «основной программы» функционального блока, пересчитываемой циклически при циклическом его вызове, методы, являясь *функциями*, вызываются «однократно» и должны полностью выполнить свою задачу за один вызов. Например, если все условия для включения аппарата созданы, метод TurnOn просто переводит его из состояния «выключен» в состояние «включается», формирует соответствующее сообщение и возвращает TRUE. Вся дальнейшая работа возлагается на «основную программу». Если в данный момент включить аппарат не представляется возможным, формируется сообщение об этом и возвращается FALSE. Методы TurnOff и Reset работают аналогично.

Таким образом, в распоряжении программиста оказывается мир, населенный достаточно «умными» сущностями, «знающими» друг о друге, умеющими «общаться» между собой и безопасно выполнять приказы (или отказывающимися их выполнять). Реализация плана переключений в таких условиях не представляет особых проблем.

Реализация плана

План переключений можно представить массивом структур, каждая из которых описывает какое-то действие. В простейшем случае структура может содержать ссылку на аппарат и указание необходимого действия:

```
TYPE OneAction:
STRUCT
```

```
        Apparat: Device;
        Act:BOOL; //0 - выкл, 1 - вкл
END_STRUCT
END_TYPE
```

Пример плана:

```
PlanLineOn: ARRAY[0..2] OF OneAction := [
(Apparat:=GVL.GK_0155, Act:=FALSE),
(Apparat:=GVL.DC_0155, Act:=TRUE),
(Apparat:=GVL.HVCB_0155, Act:=TRUE)];
```

Здесь GK_0155, DC_0155, HVCB_0155 – экземпляры функциональных блоков GK, DC, и HVCB соответственно, объявленные в глобальном списке GVL и проинициализированные в программе Initialization.

Выполнение плана можно поручить специальному функциональному блоку. В упрощенном виде (без контроля времени) показаны его секции объявления и кода:

```
FUNCTION_BLOCK PlanExecutor
VAR_INPUT
        IN:BOOL:=FALSE; //запуск
//указатель на первую структуру плана
        PlanLine: POINTER TO OneAction;
        //число шагов
nsteps:BYTE;
END_VAR
VAR_OUTPUT
        Done:BOOL:=FALSE; //план выполнен
END_VAR
VAR
        i:BYTE:=0; //счетчик
END_VAR
```

```
IF NOT IN THEN
        i:=0; Done:=FALSE; RETURN;
END_IF
IF i=nsteps THEN
        Done:=TRUE; RETURN;
END_IF
IF PlanLine[i].Act AND PlanLine[i].Apparat.State = 'off' THEN
        PlanLine[i].Apparat.TurnOn();
ELSIF NOT PlanLine[i].Act AND PlanLine[i].Apparat.State = 'on' THEN
        PlanLine[i].Apparat.TurnOff();
ELSIF (PlanLine[i].Act AND PlanLine[i].Apparat.State = 'on') OR
         (NOT PlanLine[i].Act AND PlanLine[i].Apparat.State = 'off')
THEN
        i:=i+1;
END_IF
```

Объявление экземпляра и вызов его на исполнение:

```
PE:PlanExecutor;
```

PE(IN:=TRUE,PlanLine:=ADR(PlanLineOn),nsteps:=3);

Прототип программы был реализован в CODESYS 3.5 и опробован на виртуальном контроллере CODESYS Control Win V3. На рис. 2 показано окно визуализации, с помощью которого проводилась отладка. Система позволяет осуществлять независимое управление аппаратами посредством кнопок, расположенных слева, а также запускать на выполнение программы переключений (включение и выключение линии в целом) с помощью кнопок в верхнем правом углу окна. Сигнальные лампы привязаны к дискретным «выходам» управления приводами и загораются при включении приводов. Тумблеры привязаны к дискретным «входам» и предназначены для формирования (вручную) «подтверждающих сигналов» о срабатывании концевых выключателей приводов.

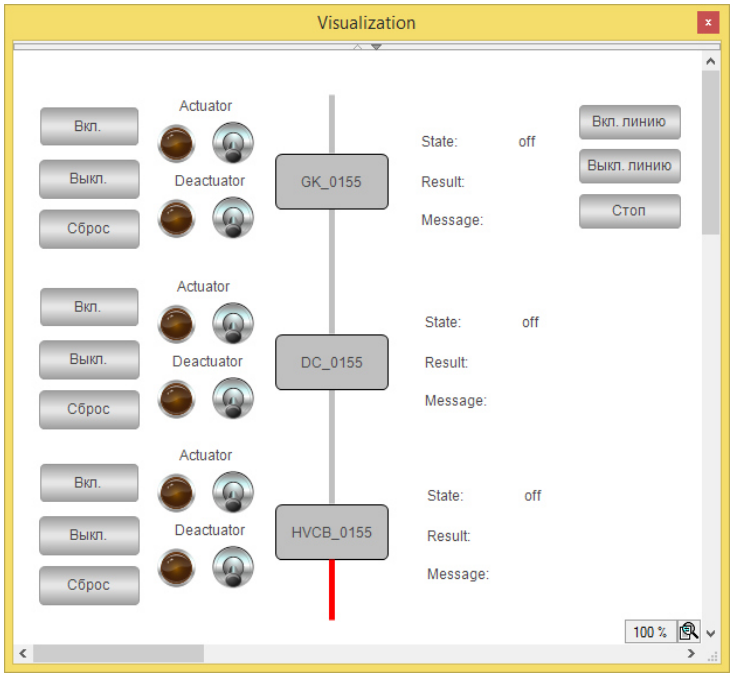

Рис. 2. Пользовательский интерфейс для отладки.

Для каждого аппарата выводится информация о его состоянии, результате последнего вызова методов TurnOn и TurnOff, а также последнее сообщение, которое было им сгенерировано.

Заключение

В результате апробации программного прототипа выявлена его работоспособность. Подсистема формирования программы переключения с графическим интерфейсом пользователя была доработана и сопряжена с исполнительной подсистемой выпускником кафедры автоматизации производственных процессов и электротехники Амурского государственного университета Р.Е. Михолап при выполнении им выпускной квалификационной работы. В качестве объекта управления использованы элементы реальной схемы распределительного устройства 220 кВ подстанции Хани.

1. Кузьмина, Н. Объектно-ориентированное программирование в стандарте МЭК 61131-3 // Современные технологии автоматизации. – 2017. – №4. – С.104-110.

2. CODESYS Group. Официальный сайт [Электронный ресурс]. – Режим доступа: https://www.codesys.com/. – 10.04.2023.

3. Контрольно-измерительные приборы ОВЕН: датчики, контроллеры, регуляторы, измерители, блоки питания и терморегуляторы. Официальный сайт компании ОВЕН [Электронный ресурс]. – Режим доступа: https://owen.ru/ . – 10.04.2023.

4. ГОСТ Р МЭК 61131-3-2016 Контроллеры программируемые. – Часть 3. Языки программирования [Электронный ресурс]. – Режим доступа: https://docs.cntd.ru/. – 10.04.2023.