К.В. ВАВИЛОВ (ООО "НИИЭФА-ЭНЕРГО"),
А.А. ШАЛЫТО (СПбГУ ИТМО)

Что плохого в неавтоматном подходе к программированию контроллеров?

Изложены проблемы, возникающие при традиционном подходе к реализации последовательностной логики в контроллерах, в которых программный цикл организован в виде последовательного выполнения команд. В качестве эффективного решения этих проблем предложено применять автоматный подход (технологию автоматного программирования или SWITCH-технологию). Этот подход, в частности, не только позволяет обеспечить полное соответствие алгоритма и кода программы, реализующего алгоритм, но и возможность автоматической генерации кода.

This article investigates the problems of a traditional approach to the implementation of sequential logics in controllers with the programming cycle organized as a successive command execution. An efficient solution to these problems is suggested using the automata-based approach (automata-based programming technology or SWITCH-technology). This approach ensures the complete conformity of the algorithm and its source code, and provides the possibility of automatic code generation.

Особенности выполнения программы контроллера

В качестве примера рассмотрим программируемые логические контроллеры серии *SIMATIC S7-200* фирмы *Siemens* (в дальнейшем – контроллеры *S7-200*).

Центральный процессор (CPU) контроллера *S7-200* предназначен для циклического выполнения ряда заданий, включая программу пользователя. Такая циклическая работа процессора называется циклом сканирования. В течение цикла сканирования CPU выполняет все или большинство из следующих задач (рис. 1):

- считывание значений входов;
- выполнение программы;
- обработку коммуникационных запросов;
- выполнение самодиагностики CPU;
- запись в выходы.

Важные моменты, которые необходимо учитывать при реализации логики:

- каждый цикл сканирования начинается со считывания текущих значений цифровых входов и последующей записи этих значений в регистр входов образа процесса;

Рис. 1. Цикл сканирования контроллера

- CPU *реализует программу последовательно*, начиная с первой команды и до последней;
- в конце каждого цикла сканирования CPU записывает значения, хранимые в регистре выходов образа процесса, в цифровые выходы.

Языки программирования

В качестве языка программирования производителем контроллера рассматриваемого типа предлагается использовать ассемблероподобный текстовый язык *STL* и его графические аналоги – *LAD* и *FBD*. Программировать на таких языках из-за низкого уровня абстракции весьма трудно, тем более что производителем никаких подходов к программированию, отличных от традиционного (бери систему операторов и пиши программу), не предлагается.

Что плохого в традиционном подходе?

Приводимый пример поможет понять минусы традиционного (неавтоматного) подхода к алгоритмизации управления для контроллеров.

Пусть имеется техническое задание.

Человек, находясь на одном этаже, должен проехать на лифте на другой этаж. Система управления должна реализовать следующую последовательность действий:

- человек нажимает кнопку вызова лифта;
- когда двери подъехавшего лифта открываются, человек входит в лифт;
- человек нажимает кнопку нужного этажа;
- когда лифт остановился на заданном этаже и двери открылись, человек выходит из лифта.

Приведенная в ТЗ последовательность действий (или предложенный заказчиком алгоритм решения задачи) может быть изображена в виде схемы алгоритма, представленной на рис. 2.

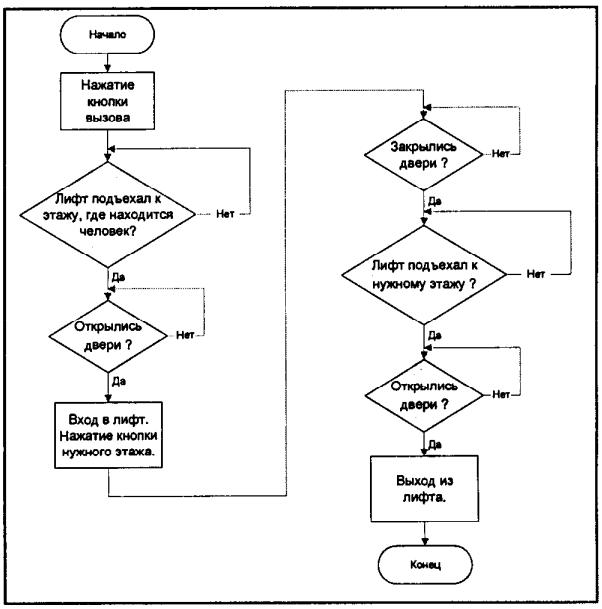

Рис. 2. Исходная схема алгоритма

Это плохое решение. Поясним почему. Данный алгоритм содержит циклы ожидания того или иного события.

Теперь достаточно вспомнить, как выполняется цикл сканирования CPU, и станет ясно, что *нельзя логически зацикливать программу*, хотя бы потому, что тогда контроллером *не будет производиться* ввод информации, а без информации с входов контроллера нельзя выйти из логического цикла.

Традиционно эта проблема имеет несколько решений.

Используются функции непосредственного чтения входов контроллера.

Используется механизм прерываний. Например, программируется опрос входов через определенные промежутки времени.

Эти решения имеют следующие недостатки.

Такое построение программы контроллера никак не отражено в алгоритме.

Существует вероятность некоторой асинхронности выполнения разных частей программы, так как нарушается строгая последовательность "ввод – обработка – вывод".

Приведенный алгоритм достаточно прост. Если представить более сложную логику, где данный алгоритм является вызываемым, то необходимо учитывать события, ожидаемые вызывающим алгоритмом.

Однако приведенные недостатки почему-то никого не "пугают". И это несмотря на то, что если необходимо выполнять параллельно и независимо несколько (два и более) подобных алгоритмов, то такая задача при традиционном подходе оказывается практически нереализуемой.

Выход из сложившейся ситуации все-таки есть. Для того чтобы предотвратить логическое зацикливание, вероятнее всего (и, похоже, это единственный вариант решения) придется фиксировать некие стадии (этапы), которые программа уже прошла. Преобразованная схема алгоритма приведена на рис. 3.

Алгоритм на рис. 2 представляет собой формализованное описание последовательностной логики, соответствующее ТЗ, и не учитываст особенностей реализации. Алгоритм на рис. 3 по существу является алгоритмом реализации в конкретной среде выполнения с учетом ее особенностей. Он также соответствует техническому заданию, но дополнен необходимой информацией для реализации в указанной среде алгоритма, представленного на рис. 2.

При этом все-таки не учтено следующее. В ТЗ практически никогда не описываются временные (от слова "время") особенности процесса управления. В частности, в рассматриваемом примере описывается нормальная работа лифта (нормальная реакция на действия человека). Считается, что когда-то (за конечное время) лифт подъедет к этажу, когда-то откроются двери, когда-то лифт доедет до заданного этажа и т.д.

Первое, что придется сделать дополнительно к алгоритму на рис. 3, это добавить контроль времени к каждому пока "бесконечному" ожиданию события. Если после истечения времени контроля событие не произошло, необходимо определить (и согласовать с заказчиком) действия системы при этом. Однако и это далеко не все. Во время ожидания могут произойти события, нарушающие нормальный ход процесса, например, после нажатия кнопки лифт не поехал, лифт остановился, не доехав до требуемого этажа, заклинило двери.

Выход из этих ситуаций на разных стадиях алгоритма производится по-разному. Таким образом, алгоритм должен быть дополнен "нехорошими" условиями и, соответственно, действиями для выхода из "нехороших" ситуаций. Это отразить на схеме алгоритма, сильно не переделывая его, весьма непросто. Теперь вроде бы все учтено. Осталось отобразить новый алгоритм и заняться написанием программы по нему, благо в алгоритме заложено практически все необходимое для реализации.

Такая схема алгоритма обычно очень громоздка, и поэтому она на практике почти никогда не используется: схему алгоритма один раз еще можно построить, но учесть в ней *все* особенности реального алгоритма, а тем более вносить в нее изменения на всех этапах жизненного цикла программы, не будет практически *никто*. Это связано с тем, что такая схема не только плохо "читается" (из-за размеров и, например, наличия огромного числа пересекающихся линий), но и из-за сложности ее анализа. Приходится учитывать промежуточную информацию, существующую только для того, чтобы обеспечить корректность реализации конкретным методом (в данном алгоритме это признаки

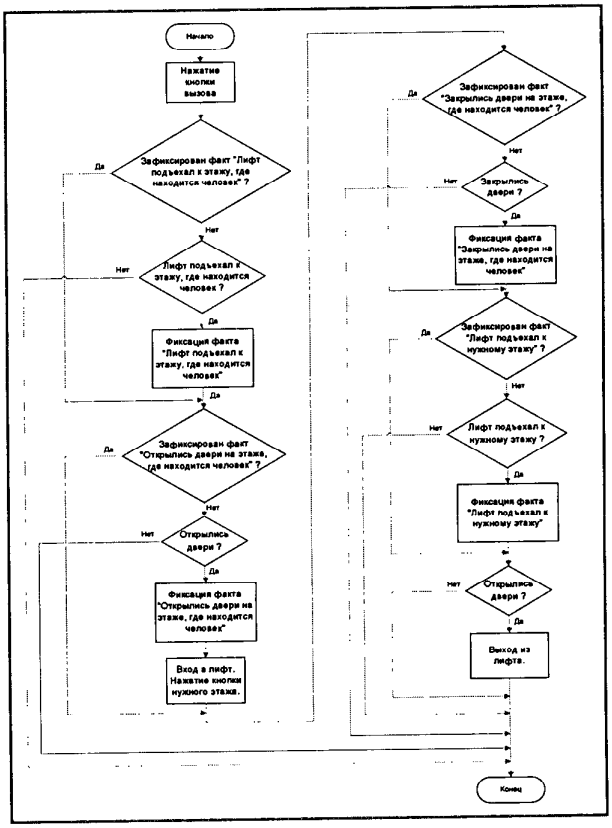

Рис. 3. Преобразованная схема алгоритма

фиксации прошедшей стадии). И это еще без учета собственно программирования, которое придется производить на не очень удобном ассемблероподобном языке, который по сути только и есть у контроллера *S7-200*.

Таким образом, если для реального проекта полноценную схему алгоритма еще кое-как и можно один раз построить, то вносить в нее изменения не будет никто, а корректироваться будет только программа, и поэтому схема алгоритма и программа "рассинхронизируются", и корректную проектную документацию при таком подходе не получить. Указанные особенности применения схем алгоритмов приводят к тому, что их на практике либо не строят, либо не включают в проектную документацию.

Таково положение при использовании традиционного подхода к алгоритмизации и программированию контроллеров [1].

Итак, имеются следующие предпосылки для применения другого подхода.

- *Явные сложности алгоритмизации традиционным способом. Следствием этого является возможность создания только некоторого "обобщенного" алгоритма, в котором не учитываются многие детали.*
- *Несоответствие алгоритма, созданного традиционным способом, и кода программы.*
- *Трудоемкость "ручного" программирования на ассемблероподобном языке, особенно в отсутствие полноценной и корректной модели программы.*
- *Недокументированность решений, принимаемых программистом.*

Почему целесообразно применять автоматный подход

Возвращаясь к схеме алгоритма на рис. 3, отметим, что *фиксация прошедшей стадии алгоритма есть ничто иное, как явное введение состояний.* Остается применить автоматный подход [2], перерисовав алгоритм в виде графа переходов, который может быть изображен существенно более компактно по сравнению со схемой алгоритма. Этот граф формально и изоморфно реализуется в виде текста программы даже на языках низкого уровня (глава 14 в [2]). Построение графа переходов может быть выполнено эвристически по ТЗ или даже формально по схеме алгоритма [3], если такая полноценная схема имеется.

Перечислим задачи, решаемые при применении автоматного подхода для программирования контроллеров.

1. Создание алгоритма, учитывающего особенности выполнения программы контроллера (в частности, последовательное выполнение операций процессором и невозможность логического зацикливания).

2. Создание алгоритма в виде, наиболее близком к реализации. Это позволяет свести программирование к рутинной работе и, в частности, автоматизировать процесс построения кода программы по графам переходов.

3. Создание алгоритма в виде, пригодном для анализа людьми, например, технологами, которые "далеки" от программирования. Это также полезно и разработчикам, так как гораздо проще разобраться в логике, изображенной в виде диаграммы на бумаге, чем изучать программный код, особенно такой, как у рассматриваемого контроллера.

4. Создание алгоритма в компактной форме.

Алгоритм, спроектированный на принципах автоматного программирования [4], а что самое важное, *программа, созданная на основе этого алгоритма,* полностью отвечают вышеизложенным требованиям.

Разработанная авторами методика и конвертор "Схема алгоритма – код программы" позволяют реализовать задачи логического управления любой сложности [5], причем с повышением сложности задачи эффективность применения данного подхода возрастает.

Более подробно с автоматным подходом и примерами его применения можно ознакомиться на сайте http://is.ifmo.ru. Этот подход может быть применен не только для контроллеров, но и других аппаратных и программных средств, как, например, описано в работе [6]. Автоматный подход в программировании контроллеров все шире используется и не только в ближайшем окружении авторов [7].

Константин Валерьевич Вавилов – начальник группы АСУ, ООО "НИИЭФА-ЭНЕРГО".
Телефон (812) 464-66-80 (доб. 238).
E-mail: wawi@yandex.ru
Анатолий Абрамович Шалыто – д-р техн. наук, профессор, заведующий кафедрой "Технологии программирования" Санкт-Петербургского государственного университета информационных технологий, механики и оптики.
Телефон (812) 702-55-45.
E-mail: shalyto@mail.ifmo.ru

Список литературы

1. *Шалыто А.А.* Использование граф-схем и графов переходов при программной реализации алгоритмов логического управления // Автоматика и телемеханика. 1996. № 6 (http://is.ifmo.ru/works/gsgp1_/1/), № 7 (http://is.ifmo.ru/works/gsgp2_/1/).
2. *Шалыто А.А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. (http://is.ifmo.ru/books/switch/1).
3. *Туккель Н.И., Шалыто А.А.* Преобразование итеративных алгоритмов в автоматные // Программирование. 2002. № 5. (http://is.ifmo.ru/works/iter/).
4. *Шалыто А.А.* Технология автоматного программирования // Мир ПК. 2003. № 10. (http://is.ifmo.ru/works/tech_aut_prog/).
5. *Вавилов К.В.* Программируемые логические контроллеры *SIMATIC S7-200* (*Siemens*). Методика алгоритмизации и программирования задач логического управления. (http://is.ifmo.ru/progeny/_metod065.pdf).
6. *Вавилов К.В., Шалыто А.А.* LabVIEW и SWITCH-технология // Промышленные АСУ и контроллеры. 2006. № 6. (http://is.ifmo.ru/progeny/_vavilov2.pdf.zip).
7. *Петров И.В., Вагнер Р.* Отладка прикладных ПЛК программ в *CoDeSys* (часть 3) // Промышленные АСУ и контроллеры. 2006. № 4. (http://www.3s-software.com/se_data/_filebank/press/2006/iecdeb03.pdf).