SWITCH-ТЕХНОЛОГИЯ В ЗАДАЧАХ ЛОГИЧЕСКОГО УПРАВЛЕНИЯ

В.А. Татарчевский

Для построения систем *логического управления* (ЛУ) широко используются промышленные компьютеры и *программируемые логические контроллеры* (ПЛК). При проектировании систем ЛУ основная доля трудоемкости приходится на разработку *программного обеспечения* (ПО). От надежности функционирования ПО зависит надежность работы всей системы. Однако построение надежного ПО представляет собой сложную задачу. На надежность ПО влияют уровень квалификации программистов, методы проектирования и тестирования ПО, применяемые в процессе разработки [1,2]. В последнее время появились продукты и методики для повышения эффективности разработки ПО (*IBM Rational Rose* [3] и метод экстремального программирования [4]), однако в силу различных причин ни один из этих методов не обеспечивает решения всех проблем, возникающих при разработке ПО.

Одной из главных проблем разработки ПО является недокументированность (или слабая документированность) программных проектов [5,6]. В большинстве случаев основным документом, описывающим структуру и поведение программы, является ее исходный текст, что сильно осложняет процессы отладки и сопровождения ПО. Некоторые надежды на изменение ситуации связаны с появлением графического языка *UML* [7,8], мощного инструмента описания структуры и поведения ПО.

Для задач ЛУ особое значение имеет раздел *UML*, описывающий диаграммы состояний, представляющие собой, по сути, графы переходов *конечного автомата* (КА) с некоторыми отличиями. Они заключаются в том, что состояния являются сложными объектами, имеющими в общем случае действие при входе, действие при выходе, деятельность, внутренние переходы и отложенные события. Также возможны составные состояния, имеющие подсостояния, и параллельные потоки. В рамках модели диаграмм состояний *UML* можно описывать автоматы Мура, Мили, а также автоматы смешанного типа.

Главное преимущество применения диаграмм состояний при проектировании систем ЛУ заключается в том, что появляется возможность создания графической документации на ПО, точно и в полной мере отражающей поведение и структуру ПО. Представление логики программы в виде КА позволяет применить для анализа поведения ПО теорию КА. При этом КА может быть исследован на предмет достижимости всех его состояний, отсутствие тупиковых состояний, может быть проанализирована правильность взаимодействий частей программы.

Основным методом реализации программ, представленных в виде КА на *языке высокого уровня* (ЯВУ), является SWITCH-технология, называемая также программированием с явным выделением состояний, или автоматным программированием. Она получила свое название от оператора *switch* языка *C*, который является основной структурой при написании программ в SWITCH-технологии.

SWITCH-технология (см. *http://is.ifmo.ru*) разрабатывается в России с 1991 года. Ведущая роль в разработке технологии принадлежит А.А. Шалыто, профессору Санкт-Петербургского государственного университета информационных технологий, механики и оптики [9,10–12].

В данной статье рассмотрен способ повышения эффективности SWITCH-технологии, основанный на введении в модель КА таймеров и параллельных потоков в явном виде.

Таймером в предлагаемой модели является выделенная переменная, увеличивающая свое значение через фиксированные интервалы времени (например, каждую секунду). При входе в состояние значение этой переменной обнуляется (таймер инициализируется). В результате появляется возможность указывать в качестве условия выхода из состояния определенное значение данной переменной, соответствующее необходимому интервалу времени. Это позволяет упростить граф переходов КА.

Для задач ЛУ характерно наличие параллелизма. *UML* описывает только один вариант параллелизма в диаграммах состояний, заключающийся в том, что составное состояние может включать в себя несколько КА, работающих параллельно. В данной статье предлагается указывать параллельные потоки на графе переходов в явном виде, используя для изображения разветвлений и слияний потока управления специальный символ из двух параллельных линий.

Рассмотрим пример реализации автомата, использующего потоки и таймеры [9] (рис. 1 и 2). Опишем работу системы (рис. 1).

1. В исходном состоянии все клапаны закрыты, двигатель мешалки выключен.

2. Этап 1. При нажатии кнопки «Пуск» производится налив жидкости путем открытия клапана 1 до срабатывания датчика **X3**.

3. Этап 2. Производится налив жидкости путем открывания клапана 2 до срабатывания датчика **X4**.

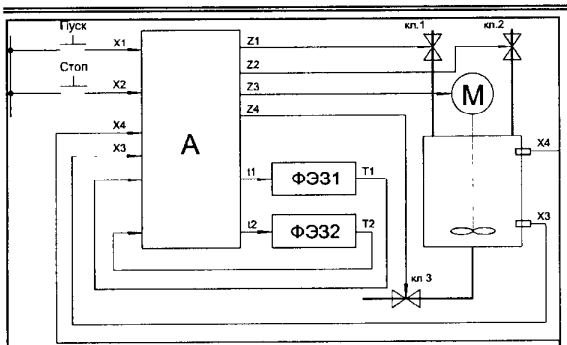

Рис. 1. Схема автоматизации: ФЭЗ – функциональный элемент задержки, М – мотор, кл. – клапан

4. Этап 3. Включается двигатель мешалки на время **t1**.

5. Этап 4. Производится слив смеси путем открывания клапана 3 на время **t2**.

6. После выполнения пункта 5 система возвращается в исходное состояние.

7. При нажатии кнопки «Стоп» во время выполнения пунктов 2 – 4 система переходит к сливу смеси (пункт 5).

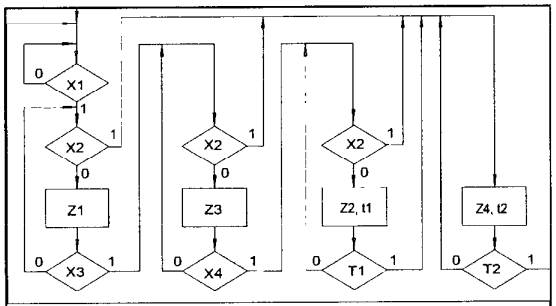

Рис. 2. Схема алгоритма работы системы управления

Построим по данному описанию автомат Мура, использующий таймеры (рис. 3).

Заметим, что схема алгоритма (рис. 2) содержит 12 элементов, а эквивалентный ей граф переходов на рисунке 3 состоит всего из пяти состояний. При этом за счет введения таймера (одной переменной на весь автомат) мы избавились от «искусственных» входов **T1**, **T2**, выходов **t1**, **t2** и двух элементов задержки **ФЭЗ1** и **ФЭЗ2**, которые были на схеме автоматизации (рис. 1). Но у графа переходов есть недостаток – из всех «рабочих» состояний, кроме последнего (из состояний 2–4), в состояние 5 ведут переходы с условием **X2**, (нажатие кнопки «Стоп»). С учетом того, что в сложных технологических установках число состояний автомата в рабочем цикле может достигать нескольких сотен, такие переходы сильно загромождают схему. Также подобный «прямой» подход может привести к ошибкам и при разработке программы, и при ее модификации. Теперь представим себе, что в процессе разработки было решено использовать в качестве кнопки «Стоп» не нормально-разомкнутую, а нормально-замкнутую. Для этого необходимо произвести в соответствующих переходах из состояний 2–4 замену **X2** на **~X2** (**X2** инверсное). Это небольшое изменение является ошибкоопасным, так как замену придется производить во всех рабочих состояниях. Поэтому целесообразно преобразовать граф переходов на рисунке 3 в схему с параллельными потоками, изображенную на рисунке 4.

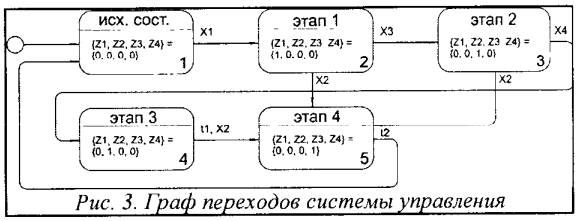

Рис. 3. Граф переходов системы управления

Несмотря на то, что количество состояний в новой схеме увеличилось до шести, число переходов в ней уменьшилось, а ее наглядность и модифицируемость возросли. Введем любое количество состояний рабочего цикла вместо состояний 2–4, не заботясь о корректности срабатывания кнопки «Стоп», а замена условия останова системы (например, замена **X2** на **~X2**) влечет за собой изменение только в одном месте программы.

Сравним реализацию данного алгоритма на языке *C* при традиционном подходе и при использовании SWITCH-технологии. Код обоих вариантов приведен в таблице, в левом столбце таблицы – код для схемы алгоритма по рисунку 2, в правом – для схемы, приведенной на рисунке 4. Следует отметить, что в таблице отображена скорее условная реализация, чем реальный код, в нем опущены многие подробности реализации, такие как объявления переменных и функций, опрос входов автомата и выдача воздействий на выходы. Функция *starttimer()*, обнуляющая таймерную переменную, должна выполняться только при входе в состояние, что довольно просто решается введением дополнительного флага.

Реализация алгоритма ЛУ при традиционном подходе и в SWITCH-технологии

```
//Традиционный подход
while(1) {
  z1 = z2 = z3 = z4 = 0;
  while(~x1);
  if(~x2){
    z1 = 1;
    while(~(x3 || x2));
    z1 = 0;
    if(~x2){
      z3 = 1;
      while(~(x4 || x2));
      z3 = 0;
      if(~x2){
        x2 = 1; t1 = 1;
        while(~(T1 || x2));
        x2 = 0; t1 = 0;
      };
    };
  };
  z4 = 1; t2 = 1;
  while(~T2);
};
```

```
//switch-подход
while(1) {
  switch(s){ //s - состояние
    case1: z1 = z2 = z3 = z4
= 0;
      if(x1) s = 2;
      break;
    case2: z1 = 1;
           z2 = z3 = z4 = 0;
      if(x3) s = 3;
      break;
    case3: z1 = z2 = z4 = 0;
           z3 = 1;
      if(x4) s = 4;
      break;
    case4: z1 = z3 = z4 = 0;
           z2 = 1;
           starttimer();
      if(timer >= t_1) s = 5;
      break;
    case5: z1 = z2 = z3 = 0;
           z4 = 1;
           starttimer();
      if(timer >= t_2) s = 1;
      break;
  };
  if((s > 1)&&(s < 5)&& x2)
    s = 5;
};
```

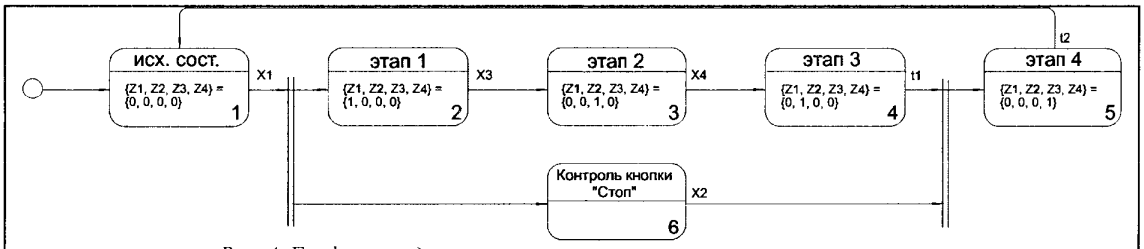

Рис. 4. Граф переходов системы управления с параллельными потоками

Реализация алгоритма, записанного в виде схемы алгоритма неочевидна. Даже для столь простой программы она требует от программиста некоторых творческих усилий, интуитивного подхода к написанию текста программы. Алгоритм может быть описан на ЯВУ разными способами, и выбор одного из них является делом личных предпочтений программиста, его интуиции, квалификации и опыта. Внесение в программу даже небольших изменений сопряжено с трудностями и высокой вероятностью внесения ошибок в ПО.

Программа, построенная на основе SWITCH-технологии, имеет больший объем, но меньшую *структурную сложность*. Ее код изоморфен графу переходов. Если сравнивать затраты времени в том и другом случае, то выигрыш от применения SWITCH-технологии очевиден.

Процесс перевода графа переходов в текст на ЯВУ является абсолютно формальным. Надо только создать шаблон программы с функциями ввода-вывода и таймеров, тогда можно легко написать программу с весьма большим числом состояний. При этом нетрудно проверить соответствие текста программы графу переходов. Более того, в силу формальности преобразования процесс кодирования можно полностью автоматизировать, разработав компилятор графов переходов, тем самым исключив человеческий фактор из процесса кодирования. Примером такого компилятора может служить инструментальное средство *Unimod*, разрабатываемое российской компанией «eVelopers» [13].

Если граф переходов корректен, то этап отладки при таком методе проектирования отсутствует. Отметим, что при традиционных методах программирования отладка занимает до 50% времени разработки [2]. Представим, что ошибка вкралась в граф переходов и осталась незамеченной вплоть до этапа тестирования. При испытаниях системы выяснилось, что программа зависает или производит неправильные действия на одном из этапов. Поиск ошибки в данном случае сводится к выводу на дисплей *номера состояния*, то есть локализация места ошибки происходит мгновенно.

Важнейшим свойством SWITCH-технологии является ее принципиальная многозадачность, не требующая при этом применения многозадачной операционной системы (в микроконтроллерных системах с ограниченными ресурсами) и ОС как таковой. При этом возможно одновременное исполнение и взаимодействие множества КА.

В заключение можно отметить, что в статье рассмотрены основы применения SWITCH-технологии в системах ЛУ, показаны преимущества данной технологии, заключающиеся в значительном сокращении сроков разработки, сокращении (или вовсе исключении) этапа отладки, достижении полного соответствия программы программной документации и техническому заданию, в повышении модифицируемости ПО. Главное же преимущество SWITCH-технологии состоит в переходе от интуитивных методов разработки к формальным, от вероятностных методов оценки надежности ПО к доказательству надежности ПО, основанному на структуре его графов переходов.

Автор выражает глубокую благодарность Анатолию Абрамовичу Шалыто за участие в подготовке статьи.

Список литературы

1. Константин Л. Человеческий фактор в программировании. - СПб.: Символ-Плюс, 2004. - 384 с.

2. Брукс Ф. Мифический человеко-месяц, или как создаются программные системы.- СПб.: Символ-Плюс, 2005.-304 с.

3. Боггс У., Боггс М. UML и Rational Rose. - М.: Лори, 2001. - 608 с.

4. Бек К. Экстремальное программирование. Библиотека программиста. - СПб.: Питер, 2002. - 224 с.

5. Шалыто А.А. Новая инициатива в программировании. Движение за открытую проектную документацию //Мир компьютерной автоматизации. - 2003. - № 5. - С.67–71.

6. Шалыто А.А. Еще раз об открытой проектной документации //PC Week/RE. - 2005. - № 11, - С. 33-34.

7. Фаулер М. UML. - СПб.: Символ-Плюс, 2004. - 192 с.

8. Буч Г., Рамбо Д., Джекобсон А. Язык UML. Руководство пользователя. - М.: ДМК Пресс. - СПб.: Питер, 2004. - 432 с.

9. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. - СПб.: Наука, 1998. 628 с.

10. Шалыто А.А. Логическое управление. Методы аппаратной и программной реализации. - СПб.: Наука, 2000. - 780 с.

11. Шалыто А.А. Алгоритмизация и программирование задач логического управления. - СПб.: СПбГУ ИТМО. - 1998. - 56 с.

12. Шалыто А.А. Использование граф-схем и графов переходов при программной реализации алгоритмов логического управления //Автоматика и телемеханика. - 1996, - № 6. - С. 148 – 158; № 7. - С. 144–169.

13. Гуров В., Нарвский А., Шалыто А. Исполняемый UML из России //PC Week/RE. - 2005. - № 26. - С. 18-19.