УДК 519.683.8: 681.5

В.Н. Негода

УНИФИКАЦИЯ ПРОЕКТНЫХ РЕШЕНИЙ ПРИ АВТОМАТНОМ ПРОГРАММИРОВАНИИ СИСТЕМ ЛОГИЧЕСКОГО УПРАВЛЕНИЯ

Негода Виктор Николаевич, *доктор технических наук, окончил радиотехнический факультет Ульяновского политехнического института, профессор кафедры «Вычислительная техника» Ульяновского государственного технического университета. Имеет статьи, монографии и авторские свидетельства в области проектирования встроенных систем контроля и управления. Область научных интересов – автоматизация проектирования логического управления техническими системами. [e-mail: nvn@ulstu.ru].*

Аннотация

Для повышения технологичности процесса автоматного программирования систем логического управления предлагается использовать унифицированный набор проектных решений. Этот набор охватывает такие представления состояний, функций переходов и выходов формальной модели конечного автомата, которые учитывают требования реального времени при создании распределенных систем управления. Унификация позволяет создавать устойчивые шаблоны проектирования, классы программных объектов и программные функции с высокой степенью повторности использования. Рассматриваются аспекты применения унифицированных проектных решений в системах автоматической генерации кода программ логического управления и связанных с ними сервисных программ.

Ключевые слова: распределенные системы управления, проектирование систем логического управления, автоматное программирование, шаблоны проектирования, автоматическая генерация программ.

THE UNIFICATION OF DESIGN SOLUTIONS AT AUTOMATA-BASED PROGRAMMING OF LOGIC CONTROL SYSTEMS

Viktor Nikolaevich Negoda, *Doctor of Engineering; graduated from the Faculty of Radio-Engineering of Ulyanovsk Polytechnic Institute; Professor at the Department of Computer Science of Ulyanovsk State Technical University; an author of articles in the field of computer-aided design of embedded systems; area of expertise – computer-aided design of technical systems with logical control. e-mail: nvn@ulstu.ru.*

Abstract

The unified set of design solutions are proposed for the efficient automata-based programming of the logic control systems. This set includes such viewing of machine state, transition and output of formal finite state machine model, that take real-time requirements into consideration while creating distributed control systems. Unification allows to establish design patterns and create reusable classes of software objects and functions. Different application aspects of the unified design solutions to the logic control system's source code generation are considered, together with generation of various service programs for the logic control system.

Key words: distributed control systems, logic control system design, automata-based programming, design patterns, automatic code generation.

Введение

Автоматизация проектирования программного обеспечения систем управления всегда базируется на использовании подхода, получившего в конце прошлого века название «модельно-ориентированное проектирование – Model-Based design (MBD)» [1–3], причем, активное его применение началось на несколько десятков лет ранее. Использование этого подхода при создании систем логического управления (СЛУ) привело к развитию технологии автоматного программирования [4–7]. В рамках этой технологии программы создаются на основе формальной модели конечного автомата – Finite State Machine (FSM). Наиболее распространенной формой представления проектных решений посредством FSM являются диаграммы состояний. В настоящее время разработчикам доступны десятки инструментальных средств редактирования таких диаграмм и даже поддержки моделирования поведения на основе автоматных моделей.

Важной особенностью формируемых при этом автоматных программ является существование соответствия между множеством элементов спецификации формаль-

ной модели и множеством элементов кода программы. Это соответствие позволяет существенно повысить производительность и надежность программирования благодаря сокращению семантического разрыва между проектными решениями и исходным кодом программ их реализации. Более того, соответствие позволяет автоматически генерировать исходный код автоматной программы [7, 8]. Естественно, что это делается на основе предварительно определенных формализмов:

• множества $uFSM$ всех элементов формальной спецификации автомата;

• множества $uPRG$ генерируемых элементов кода программы;

• системы отображений

$$sMap = \{sMap_i : sFSM_i \rightarrow sPRG_i \mid i \in N\}, \quad (1)$$

где $sFSM_i \subset uFSM$, $sPRG_i \subset uPRG$, N – множество чисел натурального ряда.

Практика применения автоматного программирования для создания программ логического управления технологическими процессами показывает, что указанное выше соответствие между автоматной диаграммой и исходным кодом ее реализации охватывает зачастую относительно небольшую часть целевой программы. Иначе говоря, получив в режиме автоматической генерации исходный код реализации диаграммы состояний, программист затем должен вручную дописывать то, что не охвачено формальной моделью. Причем, эти части могут по объему в несколько раз превосходить часть, полученную средствами автоматизации проектирования. Указанное обстоятельство существенно уменьшает эффект от применения таких средств, что часто приводит к отказу от их использования.

Основной причиной слабого покрытия целевого кода программы логического управления продуктами автоматической генерации, на наш взгляд, является тот факт, что автоматное программирование базируется только на логике функционирования абстрактных автоматов: Мили, Мура, С-автоматов [4]. Чтобы увеличить это покрытие, необходимо среди элементов целевых программ управления выявить такие, которые пишутся вручную и, в то же время, обладают достаточно высокой повторяемостью в различных программах, разрабатываемых с использованием автоматного подхода.

Высокую повторяемость обеспечивает унификация. Ее суть в контексте задачи проектирования СЛУ заключается в формировании наборов унифицированных проектных решений, каждый из которых обеспечивает устойчивую реализацию некоторых функций логического управления на основе автоматного подхода. Предлагается на основе некоторой части таких проектных решений модифицировать базовую формальную спецификацию FSM, расширяя ее такими элементами, которые позволяют строить генерацию кода на основе унифицированных проектных решений.

1 Базовые формальные модели и их программные реализации

Наиболее подходящей базовой моделью для автоматного программирования СЛУ технологическими процессами является смешанный автомат Мура-Мили, называемый часто С-автоматом [4]. Типовой цикл управления в СЛУ предполагает в начале цикла чтение входов, затем их обработку и в конце цикла запись выходов [6, 9]. Такому порядку действий при использовании автоматного подхода соответствует автомат второго рода, для которого функции перехода и выхода используют значения входов, считанных в текущем такте [4]. Это обстоятельство позволяет записывать функции перехода и выхода без указания автоматного времени, что соответствует такой формальной спецификации:

$$FSM = (X, Y, S, fSsx, fYs, fYsx, s0), \quad (2)$$

где X, Y – множество состояний соответственно входов и выходов автомата;

$S=\{s0, s1, ...\}$ – множество внутренних состояний автомата, которые в СЛУ обычно кодируются числами из натурального ряда N;

$fSsx$: $S \times X \rightarrow S$ – функция переходов;

fYs: $S \rightarrow Y$ – функция выходов, зависящих только от текущего состояния, т. е. функция выходов в части модели Мура;

$fYsx$: $S \times X \rightarrow Y$ – функция выходов, зависящих от текущего внутреннего состояния и состояния входов, т. е. в части модели Мили;

$s0$ – начальное состояние.

В спецификации (2) имеет место отступление от традиций использования греческих букв в обозначениях функций перехода и выхода. Это облегчает автоматизацию проектирования на стадии программирования генераторов кода. С этой же целью в именах функций фигурируют символы, характеризующие сигнатуру этих функций, а из формальной спецификации исключаются подстрочные и надстрочные символы, при этом вместо традиционной математической записи с индексом ниже используется распространенный в программировании формат обращения к элементу массива, либо даже конкатенация имени с числовым индексом.

Во многих СЛУ элементам множества X соответствуют значения бит-векторов на дискретных входах DI – Digital Input, а элементам множества Y – значения бит-векторов на дискретных выходах DO – Digital Output. Если управляющая программа должна обрабатывать значения аналоговых входов AI для ветвления в алгоритмах функционирования, то в целях унификации процессов реализации функций $fSsx$ и $fYsx$ целесообразно результатам соответствующих логических выражений ставить в соответствие разряды бит-векторов из множества X. Если нужно вычислять значения сигналов на аналоговых выходах AO на основе обработки внутренних переменных программы управления и значений входов AI, то элементы множе-

ства Y становятся неоднородными векторами, которые представляют собой композиции бит-векторов и векторов из вещественных чисел.

В автоматном программировании доминирует алгоритмическая реализация функций переходов и выходов, что предполагает их декомпозицию с целью получить логические выражения, выполняющие анализ отдельных входов и кода состояния. Однако при ориентации на табличную реализацию функций для относительно простых автоматов СЛУ спецификация (1) может применяться для генерации кода непосредственно. Унифицированное проектное решение при этом реализуется на основе выборки значений выходов и кодов состояний из таблиц функций $fSsx$, fYs, $fYsx$, формируя индекс таблицы из значений кода состояния $s \in N$ и бит-вектора $x = <x1, x2, ..>$.

Пусть, например, в СЛУ имеется nDI дискретных входа и nDO дискретных выходов, а число внутренних состояний равно nS. Очевидно, что для табличной реализации функции $fSsx$ потребуется таблица не более nS^*2^{nDI} кодов состояний. Такой же длины будет таблица для функции $fYsx$, хранящая бит-векторы Y. Таблица функции fYs будет содержать всего nS бит-векторов Y. Формальная спецификация функции $fSsx$ будет выглядеть как множество троек $<s, x, s\_new>$, где s – код текущего состояния, x – бит-вектор на входах DI, s_new – код состояния, куда осуществляется переход. Если адрес таблицы функции $fSsx$ представить как число с бит-вектором x в младших и кодом состояния s в старших разрядах, то генератор кода трансформирует множество троек в такую декларацию одномерного массива $fSsx$, в которой s_new размещается в элементе со значением индекса $(s<<nDI)\,||\,x$. Аналогично формируется таблица функции $fYsx$ на основе табличного задания соответствующей таблицы выходов. Для функции fYs адресом является код состояния s.

Программную функцию табличной реализации автомата логично строить так, чтобы она могла использоваться с различными наборами таблиц в пределах допустимых размеров бит-векторов и разрядности кода состояния. При этом нужно учесть, что сигналы на выходах должны присутствовать в течение времени. Будем считать, что для выходов задается два интервала времени: время пребывания в состоянии и время удержания выходов в фазе перехода в новое состояние. Приведенным соглашениям соответствует вариант программной реализации на языке C++ спецификации (2), представленный в листинге 1.

В этом листинге при создании объекта конструктор FSM_tabl инициирует указатели таблиц, начальное состояние, время пребывания в состоянии и время удержания выходов. Кроме того, конструктор с помощью флага autorun может запустить автомат в циклическую работу через вызов метода cycle(). Вся автоматная логика сосредоточена в функции step(), которая обеспечивает реализацию одного шага автоматного времени. В этом шаге выполняется ввод бит-вектора x, вычисление и вывод бит-вектора y в части модели Мура, вычисление и вывод бит-вектора y в части модели Мили, вычисление нового состояния state. Необходимо заметить, что по отношению ко многим дискретным выходным сигналам, которые управляют оборудованием, существуют различные требования реального времени. Учет этих требований будет обсуждаться в параграфе 3.

Традиционное автоматное программирование базируется на таком множестве элементов формальной спецификации $uFSM$, которое является спецификацией диаграммы состояний. Здесь важно заметить, что переход от классического определения С-автомата, представленного выражением (2), к автоматной диаграмме основан на двух продуктивных идеях:

а) для каждого состояния определяются такие функции перехода и выхода, которые реализуются без использования кода состояния;

```
extern int getDI(void);
extern void putDO(int);
struct FSM_tabl {
   unsigned *fSsx, *fYs, *fYsx; // адреса таблиц функций;
   unsigned nDI;      // число дискретных входов
   unsigned tstate;   // время пребывания в состоянии;
   unsigned ttrans;   // время удержания значения fYsx на выходах
   unsigned state;    // состояние
public:
   FSM_tabl(unsigned *tS, unsigned *tY, unsigned *tYx, unsigned s0,
  unsigned ts, unsigned tt, bool autorun)
   { fSsx = tS; fYs = tY; fYsx = tYx; fYsx = tYx;
      state = s0; tstate = ts; ttrans = tt;
     if(autorun) cycle();
   }
   unsigned step() // шаг автомата
   { unsigned x, ys, ysx; // вход, выходы Мура и Мили
      unsigned taddr;       // адрес таблиц функций вида f(s,x)
      x = getDI();                    // чтение с дискретных входов
      taddr = (state << x_len) || x; // вычисление адреса f(s,x)
      putDO(fYs[state]);   // реализация fY
      sleep(tstate);                  // задержка пребывания в цикле
      state = fSsx[taddr]; // переход в новое состояние
      putDO(fYsx[taddr]);             // реализация fYsx
      sleep(ttrans);                  // задержка удержания fYsx
   }
   void cycle()              // бесконечный цикл работы автомата
   { for(;;) step()}
}
```

Листинг 1. Табличная реализация автомата, заданного спецификацией (2)

это есть декомпозиция функций перехода и выхода (2), которая по сути дела базируется на обобщении разложения Шеннона, аналогичном приведенному в работе [10];

б) для каждой дуги перехода определяется логическая функция, истинность которой однозначно определяет новое состояние и значение $fYsx$.

Прежде чем привести базовую спецификацию диаграммы состояний, выполним декомпозицию функций базовой спецификации С-автомата (2), реализуя только первую из указанных идей. В результате этой декомпозиции получается система множеств функций перехода и выхода, каждая из которых не зависит от состояния s:

$$\begin{aligned}(\quad & sfSsx = \{fSx[\,s0\,](X), fSx[\,s1\,](X), \ldots\},\\ & sfYs = \{fY[\,s0\,], fY[\,s1\,], \ldots\},\\ & sfYsx = \{fYx[\,s0\,](X), fYx[\,s1\,](X), \ldots\} \qquad (3)\\ ).\end{aligned}$$

Система (3) может быть основой эффективной табличной реализации функций переходов и выходов в случаях, когда бит-вектор x имеет очень большую длину, например, многие десятки разрядов. Проектное решение, представленное в листинге 1, в этом случае не может быть использовано из-за слишком большой длины адреса таблиц $fSsx$ и $fYsx$. Для системы (3) в реальной практике проектирования СЛУ характерно, что в каждом состоянии число значимых дискретных входов в функциях $fSx[s]$ и $fYx[s]$ много меньше длины бит-вектора x. Это позволяет построить новое унифицированное проектное решение, в котором последовательность операторов функции step(), обеспечивает выбор таблиц из массивов и формирование для каждого состояния своего бит-вектора:

```
x = getDI(state);          // чтение только значимых входов
putDO(fYs[state]);         // реализация fY
      sleep(tstate);       // задержка пребывания в цикле
fSx = fSsx[state];         // определение адреса таблицы
                              fSx[s]
state = fSx[x];            // переход в новое состояние
fYs = fYsx[state];         // определение адреса таблицы
                              fYx[s]
putDO(fYx[x]);             // реализация fYx[s]
sleep(ttrans);             // задержка удержания fYsx
```

Ясно, что для приведенного выше фрагмента переменные $fSsx$ и $fYsx$ теперь должны быть указателями адресов таблиц, а не значений типа unsigned.

В практике автоматного программирования СЛУ их алгоритмы функционирования обычно изначально представляются диаграммами состояний. Эти диаграммы строит разработчик системы, возможно с участием программиста [11, 12]. Автоматная программа строится так, что каждому состоянию s соответствует фрагмент кода, в котором программно реализуется такая последовательность операций:

$$(ys = Ys;\ (PI(x)?\ (ysx = Yx1,\ s = Sx1):$$

$$P2(x)?\ (ysx = Yx2,\ s = Sx2):\ldots), \qquad (4)$$

где ys – выход автомата в части автомата Мура;

ysx – выход в части автомата Мили;

$Pi(x)$– предикат, соответствующий условию перехода из состояния s по i-й дуге;

Yxi – выход автомата в части Мили, определенный для i-й дуги;

Sxi – код состояния, куда ведет i-я дуга.

Последовательность предикатов реализуется либо условными выражениями, либо конструкциями вида:

$$if\,(P1(x))\ \{ysx{=}Yx1;\ s = Sx1\}$$
$$else\ if\,(P2(x))\ \{ysx{=}Yx2;\ s = Sx2\}\ \ldots\ .$$

В спецификации (4) представлена только одна запись, относящаяся к одному состоянию. Причем, эта запись определяет структуру элемента из областей прибытия системы отображений (1), т. е. спецификация (4) является основой для формирования множества $uPRG$.

При описании соответствующих элементов областей отправления системы отображений (1) нет необходимости что-то чему-либо присваивать. Там должны фигурировать только значения новых состояний, выходов и спецификации предикатов. Исходя из этого, формальная спецификация всей диаграммы состояний SD, где имеется множество узлов V и множество дуг U, может быть представлена следующим образом:

$$SD = \{X, Y, S, V, U, s0\}, \qquad (5)$$

где X, Y, S, $s0$ – те же элементы, что фигурируют в классической спецификации автомата(2);

$V = \{<s, Ys, Us> \mid s \in S\}$ – множество спецификаций узлов диаграммы автомата;

$Us = \{<Pj, Sxj, Yxj> \mid j \in N\}$ – множество троек, каждая из которых описывает одну дугу, выходящую из состояния s;

Pj – спецификация предиката, соответствующего условию перехода j-й дуги;

Sxj – состояние, в которое ведет j-я дуга;

Yxj – выходы в части автомата Мили, которые должны формироваться при переходе по j-й дуге.

Механизм отображения элементов спецификации (5) в элементы множества $uPRG$ достаточно прост. Элементам из X ставятся в соответствие наборы значений на входах DI, элементам из Y – наборы значений на выходах DO, а элементам из S – значения переменной s. Элементу Ys спецификации (5) ставится в соответствие присваивание $ys = Ys$ спецификации генерируемого кода (4). Элементу Pj соответствует либо логическое выражение, либо какой-то другой процесс реализации условия перехода. Если условие верно, то в части then фигурируют реализации действий, которые нужно выполнить при переходе по j-й дуге, а также оператор присваивания переменной s значения Sj, что проставлено в целевой вершине i-й дуги диаграммы.

Выбор одной из спецификаций вида (4) по коду состояния в программной реализации может осуществляться различными способами [4, 13]. Этот выбор носит характер отображения множества состояний либо в множество исполняемых фрагментов программы, либо в множество интерпретируемых спецификаций. В качестве базовых проектных решений здесь могут выступать:

1) Состояние представлено в программе явно, например, в переменной state, которая фигурирует в операторе выбора типа switch. При этом каждая ветвь оператора switch реализует одну спецификацию вида (4). Это самое распространенное решение, оно получило название switch-технологии [4, 5] и является де-факто унифицированным проектным решением, причем допускающим надежное ручное расширение кода в той части, что не покрыта автоматной диаграммой.

2) Состояние представлено в программе точкой перехода. Это так называемый метод goto [13], который приходится использовать в случаях, когда язык программирования не поддерживает оператор выбора. Это проектное решение сложнее унифицировать, поскольку его повторяемость невысока.

3) Состояние представлено явно и используется в качестве индекса массива интерпретируемых спецификаций [4, 13]. Основной проблемой унификации такого решения является сложность определения действий, не покрываемых формальной спецификацией. Кроме того, интерпретация является медленным процессом.

4) Состояние представлено неявно в виде указателя на список значений, кодирующих спецификацию (4) с использованием эффективно реализуемых представлений булевых функций. По сути дела здесь используется представление графов в виде многосвязных списков. Здесь унификации мешает малая повторяемость решения.

5) Состояние представлено объектом класса [13]. Здесь можно говорить о технологической унификации, заключающейся в использовании объектно-ориентированного и компонентного программирования.

Эффективным унифицированным проектным решением для реализации перехода по автоматной диаграмме, является введение программной функции Transit, аргументами которой являются: код состояния, куда осуществляется переход, маска установки и маска сброса разрядов бит-вектора Yx, указатель на строку текста сообщения о переходе на дисплее, указатель на строку текста сообщения о переходе в файл протокола, значение предельного времени пребывания автомата в данном состоянии. Автоматическая генерация аргументов такой функции является достаточно простой в силу того обстоятельства, что все ее аргументы в явном виде представлены на диаграмме или в спецификации (5).

2 Целеориентация и механизмы расширения базовых моделей

Причиной того, что во многих реальных проектах СЛУ использование автоматного программирования не обеспечивает формирования большей части целевого кода программы, является совокупность самых различных обстоятельств, которые рассматриваются ниже. Выявление этих обстоятельств позволяет целеориентировать выделение унифицированных проектных решений для включения их в шаблоны проектирования.

Необходимость учета реального времени. Время работы конечного автомата является дискретным, в то время как СЛУ работают в реальном времени, что означает наличие ограничений значений интервалов времени, связанных с выработкой управляющих воздействий на исполнительные механизмы и получением информации о состояниях объекта. Расширение базовых моделей, представленных спецификациями (2), (3) и (5), заключается в том, чтобы сформировать наборы переменных, в которых фиксируются ограничения реального времени и результаты измерений временных интервалов, позволяющих оценивать риски и факты нарушения соответствующих ограничений. Более детально осуществление таких расширений рассмотрено в параграфе 3.

Необходимость обеспечивать параллельную реализацию нескольких автоматных программ. В СЛУ даже средней сложности могут быть десятки процессов управления, многие из которых целесообразно реализовать средствами автоматного подхода. В большинстве случаев имеет место взаимозависимость различных процессов. Наиболее распространенный вид такой зависимости заключается в том, что некоторый основной процесс A может выполняться, только если несколько обеспечивающих процессов $B1$, $B2$, … реализуются без нарушений. Например, для процессов регулирования давления и температуры в нескольких агрегатах существуют границы допустимых значений регулируемых параметров, и устойчивое нарушение этих границ признается как условие невозможности выполнять основной процесс. Обеспечение параллелизма на различных контроллерных платформах требует различных программно-технических решений. Обеспечение выполнения ограничений реального времени при реализации нескольких автоматов существенно усложняется.

Существенное повышение степени повторности кода можно достичь, если всю программу СЛУ строить в виде совокупности объектов типа FSM. При этом выделяется главный основной автомат, функция step которого реализует основной цикл управления, и множество обеспечивающих автоматов. Множество обеспечивающих автоматов может быть разбито на подмножества, внутри каждого из которых также может существовать отношение «основной – обеспечивающие».

Эффективное управление обеспечивающими процессами может быть выполнено на базе следующих унифицированных проектных решений:

1) При кодировании состояний автоматов обеспечивающих процессов одинаковым образом нумеруются конечное состояние, состояние инициализации и аварийное состояние. Пусть эти состояния задаются начальными позициями списка имен состояний: enum {sSTOP, sINIT, sAV, ...}. Тогда инициирование процес-

сов может быть выполнено таким циклом:

```
for (i=0; i<nPA; i++)  aPA[i].setstate(sINIT);
```

где nPA – число обеспечивающих процессов;

aPA – массив объектов, каждый из которых принадлежит классу, реализующему автомат обеспечивающего процесса;

setstate – метод установки состояния в заданное значение.

Останов всех автоматов может быть выполнен аналогичным циклом, в котором выполняется функция aPA[i].setstate(sSTOP).

Проверка нормальности всех обеспечивающих процессов может быть выполнена таким фрагментом:

```
norm = true;
for (i=0; i<nPA; i++)
   norm &= aPA[i].getstate() != sAV;
```

2) Если в обеспечивающих автоматах может быть по несколько аварийных состояний, то в определение класса вводится виртуальный метод isNorm(), который для каждого автомата формирует признак нормальности обеспечивающего процесса. В этом случае в теле цикла предыдущего фрагмента будет оператор norm &= aPA[i] . isNorm() .

Взаимодействие автоматных программ может быть реализовано средствами автоматически генерируемых частей кода на основе включения в состав векторов множества Y одного автомата таких компонентов, которые обеспечивают инициирование других автоматов и формирование компонентов векторов из их множеств X.

Параллельную реализацию нескольких автоматов на основе функции step() листинга 1 можно осуществить, только если платформа поддерживает многопоточное выполнение программ. В этом случае функция sleep() обеспечивает передачу ресурсов процессора потоку, в котором может быть реализован другой автомат. В случае использования однозадачных платформ использовать задержку времени delay() вместо приостановки процесса sleep() в большинстве случаев нельзя. Всегда недопустимо использовать функцию cycle(), подобную приведенной в листинге 1. Унифицированным решением является последовательное выполнение функций step() всех автоматов, как это показано в работе [14]. Цикл последовательного вызова step() должен заменять множество функций cycle() отдельных автоматов. Для реализации задержек tstate и ttrans в автомат вводятся дополнительные состояния, в течение которых действуют заданные выходы, а выход из состояния задается по достижению времени пребывания в состоянии значений ttstate и ttrans.

Многообразие интерфейсов между системой управления и объектом. В современной практике построения СЛУ широко используются аналоговые и дискретные каналы связи, а также коммуникационные сети. Это означает, что функции getDI() и putDO() из листинга 1 становятся довольно сложно реализуемыми. В диссертации [13] достаточно детально исследованы методы автоматической генерации кода в части, зависимой от аппаратных решений интерфейсов.

Необходимость отображать ход процесса управления на устройствах индикации. Наиболее устойчивые проектные решения в части индикации излагаются ниже применительно к случаю реализации многих автоматов через объекты класса FSM.

1) Для каждого процесса выделяется свое множество элементов индикации и зон экрана контроллера. Это позволяет каждому процессу иметь независимые функции отображения.

2) В каждый объект типа FSM включается виртуальная функция отображения show(), которая вызывается либо из функции step(), либо из функции cycle(). Вызов из функции cycle() позволяет уменьшить общие затраты времени на отображение за счет того, что организуется один вызов show() на несколько вызовов step().

3) В случае организации на дисплее общего поля текстовых сообщений от большого числа автоматов, организуется менеджер сообщений, работающий над набором сообщений в каждом цикле управления. Каждый автомат любые свои сообщения помещает в очередь на обработку менеджером. В конце цикла управления менеджер выполняет селекцию сообщений в случаях, когда их слишком много для восприятия персоналом СЛУ.

Необходимость протоколировать ход процесса управления. Протоколирование может быть как в локальных устройствах памяти, так и на удаленных машинах. В качестве унифицированных решений, уменьшающих объем протоколов, время протоколирования и затраты на организацию обработки протоколов, являются:

а) использование кодов сообщений в совокупности со словарем, хранящем пары <код, текст>;

б) использование открытых форматов для представления рядов групп числовых параметров, например, формата csv.

3 Обеспечение ограничений реального времени функционирования автоматных программ

Функционирование СЛУ носит циклический характер. Наиболее общей временной характеристикой является время цикла управления Tc. При проектировании СЛУ обычно имеет место ограничение этого времени сверху: $Tc \le TcUp$.

В общем случае время цикла можно представить следующим выражением:

$$Tc = Tinet + Tiloc + Tp + Tintloc + Tintnet + Toloc + Tonet + Tshow + Tlloc + Tlnet, \quad (6)$$

где $Tinet$ – сумма времени исполнения всех функций ввода данных через коммуникационную подсистему; суммирование производится по тем функциям ввода, которые задействованы в данном цикле управления;

$Tiloc$ – сумма времени исполнения всех функций ввода данных через локальные модули, непосредственно связанные с объектом;

Tp – сумма времени исполнения всех функций обработки данных текущего цикла управления;

$Tintloc$ – сумма времени, потраченного на обработку

локальных прерываний, не связанных с реализацией цикла; в это время входит прерывание от таймера и диспетчера процессов, а также, возможно, прерываний от более высокоприоритетных процессов системы;

Tintnet – сумма времени, потраченного на обработку прерываний, связанных с выполнением запросов к серверным компонентам контроллера;

Toloc – сумма времени вывода данных через локальные интерфейсы, непосредственно связанные с объектом;

Tonet – сумма времени вывода данных в устройства объекта управления через сетевые интерфейсы;

Tshow – сумма времени исполнения функций представления процесса управления на устройствах индикации контроллера;

Tlloc – сумма времени исполнения функций протоколирования процессов управления в оперативной или флэш-памяти контроллера;

Tlnet – сумма времени исполнения функций протоколирования процессов управления на удаленных серверах.

Имеется два основных обстоятельства, снижающих точность оценки величины времени цикла Tc. Во-первых, время выполнения части программных функций, охваченных оценкой (6), существенно варьируется в зависимости от состояния объекта и самой СЛУ. Во-вторых, использование сетевой среды в условиях индустриальных помех часто приводит к потере пакетов, что заставляет повторять отправку сообщений. Это вносит большие колебания затрат времени *Tinet, Tintnet, Tonet, Tlnet*.

При создании программы СЛУ как совокупности реализаций управляющих автоматов для каждого автомата имеется пара $<Tc, TcUp>$. С целью обеспечения мониторинга значений времен Tc и локализации причин превышения верхних границ $TcUp$ в рамках автоматного подхода к программированию СЛУ целесообразно использовать следующие унифицированные проектные решения:

а) в структуру главного управляющего автомата вводятся следующие поля данных, обслуживающие слежение за реальным временем:

curTime – время начала цикла управления;

preTime – время начала предыдущего цикла;

TcMin, TcMax, TcUp – измеренные минимум и максимум времени цикла и его верхняя граница;

TinetMin, *TinetMax* – измеренные минимум и максимум времени *Tinet*;

TilocMin, TilocMax, TpMin, TpMax, TintlocMin, TintlocMax, TintnetMin, TintnetMax, TolocMin, TolocMax, TonetMin, TonetMax, TshowMin, TshowMax, TllocMin, TllocMax, TlnetMin, TlnetMax – измеренные минимумы и максимумы соответствующих времен выражения (6);

б) в структуры остальных управляющих автоматов вводятся только поля *curTime, preTime, TcMin, TcMax, TcUp*;

в) в код реализации программного цикла главного автомата помещаются функции измерения времени и фиксации всех приведенных выше значений;

г) в код реализации программного цикла главного автомата помещается функция слежения за ограничением $Tc \le TcUp$ и вывода сообщения о нарушении границы в протокол процесса;

д) в код функции step остальных автоматов помещается функция измерения и фиксации времен, а также обнаружение ограничений времени цикла и формирования сообщения о нарушении в протокол;

е) в конфигуратор программы управления вводится возможность устанавливать режим отладки, который фиксируется в соответствующем признаке;

ж) в код реализации главного автомата помещаются функции инициирования максимальных и минимальных времен и формирования отчета; первая из этих функций вызывается в режиме отладки в точках реализации таких переходов автоматной диаграммы, где начинается наиболее критическая с точки зрения обеспечения ограничений реального времени фаза технологического процесса; вторая функция вызывается также в режиме отладки в точках перехода автоматной диаграммы, где завершается критическая фаза процесса.

Благодаря приведенным проектным решениям удается в ходе отладочных и регламентных работ достаточно быстро локализовать причины возникновения нарушений реального времени и оценить степень приближения времен цикла к границам.

Одним из способов обнаружения нарушений в функционировании агрегатов объекта управления является слежение за предельным временем пребывания системы в ожидании события завершения какой-то операции. К таким операциям относится, например, перемещение до концевого выключателя. После включения привода происходит слежение за срабатыванием концевого выключателя, после чего привод должен быть выключен. Событие нарушения предельного времени называется таймаутом. Таймауты используются также в коммуникационном взаимодействии распределенных средств управления.

Унифицированное проектное решение, связанное с таймаутами, в автоматном программировании СЛУ может заключаться в следующем:

а) в структуру класса FSM вводится поле *timeOut*;

б) в реализацию перехода в состояние, где нужно фиксировать таймаут, вставляются присваивание $timeOut = curTime + timeLimit$, *где timeLimit* – предельное время пребывания автомата в данном состоянии; при использовании унифицированной функции Transit присваивание выполняется в ее коде, а один из аргументов функции задает величину *timeLimit*;

в) в начало кода реализации состояния, предусматривающего таймаут, вставляется условный оператор *if* (*curTime* > *timeOut*) *Ttransit*(*sAV, ...*); этот опера-

тор позволяет перейти к состоянию аварии, установить или сбросить выходные сигналы Y согласно алгоритму функционирования, передать сообщения на пульт оператора и в файл протокола.

Заключение

Реализация алгоритмов функционирования СЛУ средствами автоматного программирования является унифицированным проектным процессом, эффективно автоматизируемым в той части, которая опирается на автоматные модели. За пределами автоматных спецификаций оказывается много функций управления, проектирование реализаций которых практически не автоматизировано. Рассмотренные выше унифицированные проектные решения обеспечивают повышение производительности труда программиста, а в некоторых случаях расширяют пространство автоматически генерируемых программных функций целевого кода. Эти решения подтвердили свою эффективность при создании нескольких СЛУ. Всего в соответствующих проектах было создано более сотни объектов, реализованных средствами автоматного программирования.

СПИСОК ЛИТЕРАТУРЫ

1. Lennon T. Model-based design for mechatronics systems / The MathWorks Inc. – URL : http://machinedesign.com/archive/model-based-design-mechatronics-systems.

2. Ефремов А.А., Сорокин С.С., Зенков С.М. Модельно-ориентированное проектирование – международный стандарт инженерных разработок. – URL : http://matlab.ru/upload/resources/EDU%20Conf/pp%2040-43%20Sorokin.pdf.

3. Model-based design. – URL : http://en.wikipedia.org/wiki/Model-based_design#History.

4. Поликарпова Н.И., Шалыто А.А. Автоматное программирование. – СПб., 2008. – 167 с.

5. Шалыто А.А., Туккель Н.И. SWITCH-технология – автоматный подход к созданию программного обеспечения «реактивных» систем // Программирование. – 2001. – № 5. – С. 45–62.

6. Зюбин В.Е. Программирование информационно-управляющих систем на основе конечных автоматов: учеб.-метод. пособие. / Новосиб. гос. университет. – Новосибирск, 2006. – 96 с.

7. Антипова Е.В., Негода В.Н. Автоматизация проектирования программно-аппаратных реализаций автоматных диаграмм систем управления // Автоматизация процессов управления. – 2012. – № 1 (27). – С. 47–55.

8. Pavel Bekkerman. FSMGenerator. Finite State Machine generating software. – URL : http://sourceforge.net/projects/fsmgenerator/.

9. Вавилов К.В. Программируемые логические контроллеры SIMATIC S7-200(SIEMENS). Методика алгоритмизации и программирования задач логического управления. – СПб., 2005. – 68 с.

10. Горбатов А.В. Характеризационная теория синтеза функциональных декомпозиций в k-значных логиках. – М. : Изд-во физико-математ. литературы, ООО «Издательство АСТ», 2000. – 336 с.

11. Шалыто А.А. Алгоритмизация и программирование задач логического управления. – СПб. : СПбГУ ИТМО, 1998. – 55 с.

12. Harel D. Statcharts: A Visual Formalism for Complex Systems / Science of Computer Programming 8 (1987), pp. 231–274.

13. Антипова Е.В. Автоматизация проектирования аппаратно-зависимых программных реализаций автоматных диаграмм: дис. … канд. техн. наук. – Ульяновск : УлГТУ, 2012. – 211 с.

14. Негода В.Н. Специфика процессов проектирования систем контроля и управления на основе однозадачных платформ // Информатика, моделирования, автоматизация проектирования : сб. науч. тр. / под ред. Н.Н. Войта – Ульяновск : УлГТУ, 2011. – С. 17–26.

REFERENCES

1. Lennon T. *Model-based design for mechatronics systems*. The MathWorks Inc. Available at: http://machinedesign.com/archive/model-based-design-mechatronics-systems.

2. Efremov A.A., Sorokin S.S., Zenkov S.M. *Modelno-oriyentirovannoye proyektirovaniye – mezhdunarodnyy standart inzhenernykh razrabotok* [Model-based Design. Internationsl Standard of Engineering Development]. Available at: http://matlab.ru/upload/resources/EDU%20Conf/pp%2040-43%20Sorokin.pdf.

3. *Model-based Design*. Available at: http://en.wikipedia.org/wiki/Model-based_design#History.

4. Polikarpova N.I., Shalyto A.A. Avtomatnoye programmirovaniye [Automata-based Programming]. Sankt-Peterburg, 2008. 167 p.

5. Shalyto A.A., Tukkel N.I. SWITCH-tekhnologiya – avtomatnyy podkhod k sozdaniyu programmnogo obespecheniya «reaktivnykh» sistem [SWITCH-Technology: An Automated Approach to Developing Software for Reactive Systems]. *Programmirovaniye* [Programming and Computer Software], 2001, no. 5, pp. 45–62.

6. Zyubin V.E. *Programmirovaniye informatsionno-upravlyayushchikh sistem na osnove konechnykh avtomatov: ucheb.-metod. posobiye. Novosib. gos. universitet* [Programming of Information Control Systems on the base of the Finite State Automata. Tutorial. Novosibirsk State University], Novosibirsk, 2006. 96 p.

7. Antipova E.V., Negoda V.N. Avtomatizatsiya proyektirovaniya programmno-apparatnykh realizatsiy avtomatnykh diagramm sistem upravleniya [Computer-Aided Design of Software and Hardware Implementations of Control-System Automaton Diagrams]. *Avtomatizatsiya protsessov upravleniya* [Automation of Control Processes], 2012, no. 1 (27), pp. 47–55.

8. Pavel Bekkerman. *FSMGenerator. Finite State Machine generating software*. – Available at: http://sourceforge.net/projects/fsmgenerator/.

9. Vavilov K.V. *Programmiruyemyye logicheskiye kontrollery SIMATIC S7-200(SIEMENS)*. Metodika algoritmizatsii i programmirovaniya zadach logicheskogo

upravleniya [Programmable Logic Controllers SIMATIC S7-200 (SIEMENS). Procedure of Algorithmization and Programming for Logical Control Tasks]. Sankt-Peterburg, 2005. 68 p.

10. Gorbatov A.V. *Kharakterizatsionnaya teoriya sinteza funktsionalnykh dekompozitsiy v k-znachnykh logikakh* [Characterization Theory of Functional Decomposition Synthesis in k-valued Logic]. Moscow, Izd-vo fiziko-matemat. literatury, OOO Izdatelstvo AST Publ., 2000. 336 p.

11. Shalyto A.A. *Algoritmizatsiya i programmirovaniye zadach logicheskogo upravleniya* [Algorithmization and Programming for Logical Control Tasks]. Sankt-Peterburg, SPbGU ITMO Publ., 1998. 55 p.

12. Harel D. Statcharts: A Visual Formalism for Complex Systems. *Science of Computer Programming, 1987, no*. 8, pp. 231–274.

13. Antipova E.V. *Avtomatizatsiya proyektirovaniya apparatno-zavisimykh programmnykh realizatsiy avtomatnykh diagram*. Dis. kand. tekhn. nauk. [Computer-Aided Design of Hardware-Dependent Software Implementations of Automaton Diagrams. Candidate of engineering diss.]. Ulyanovsk, UlSTU Publ., 2012. 211 p.

14. Negoda V.N. Spetsifika protsessov proyektirovaniya sistem kontrolya i upravleniya na osnove odnozadachnykh platform [Scientific Features of System Design Process Monitoring and Control Systems based on the Unambiguous Platforms]. *Informatika, modelirovaniya, avtomatizatsiya proyektirovaniya:* sb. nauch. tr. pod red. N.N. Voyta [Informatics, Modeling, Computer-Aided Design. Proceedings of scientific papers under the editorship of N.N. Voyt], Ulyanovsk, UlSTU Publ., 2011, pp. 17–26.