В.Е. ЗЮБИН, канд. техн. наук,
руководитель тематической группы
“Языки технологического программирования”,
Институт автоматики и электрометрии СО РАН,
доцент кафедры “Информационно-измерительные системы”,
Новосибирский государственный университет

Пути расширения языка ST из состава МЭК 61131-3 для задач промышленной автоматизации*

В статье рассматриваются пути расширения языка *ST*, обеспечивающие его автономное использование в широком диапазоне задач промышленной автоматизации. Предложено ввести в понятийный аппарат языка концепции “процесс” и “функция-состояние”, а также модернизировать операции работы с временными объектами.

Ключевые слова: языки МЭК 61131-3, программирование алгоритмов управления, модель гиперпроцесса, процесс ориентированное программирование.

V.E. ZYUBIN

The ways to expand the ST language consisting of МЭК 61131-3 for industrial automation tasks

The article considers the ways to expand the ST language to provide its separate using over wide range of industrial automation tasks. It suggests enhancement of the language by introducing “process” and “function-condition” concepts and operations with temporal constraints.

Key words: the IEC 61131-3 languages, control algorithm programming, hyperprocess model, process oriented programming.

В 2008 г. исполнилось 15 лет со дня выхода в свет стандарта *МЭК 61131-3* [1], описывающего пять языков программирования ПЛК. 15 лет серьезный срок. По прошествии времени можно отметить, что споры вокруг этого стандарта до сих пор не утихают. Как и пятнадцать лет назад, находятся люди, использующие для программирования задач промышленной автоматизации альтернативные средства. Среди наиболее распространенных подходов следует выделить языки, базирующиеся на теории конечных автоматов, и языки системного программирования (*Cи* и *Cи++*). Причем, на рынке средств автоматизации заметны тенденции к увеличению сторонников этого подхода [2]. В последнее время в задачах автоматизации набирают популярность системы разработки, ориентированные на так называемые *софт-ПЛК* – системы, базирующиеся на технологических решениях из области ПК. В качестве примера можно вспомнить пакет *LabVIEW* [3], привлекательный не только возможностью описывать управляющие программы, но и возможностью быстрого создания интерфейса оператора для систем управления. Заметным событием на рынке систем автоматизации стало появление средства программирования робототехнических комплексов *Microsoft Robotics Studio* от компании *Microsoft*. Концепция главы *Microsoft*, озвученная в конце 2007 г. [4], означает, по крайней мере, что у стандарта *МЭК 61131-3* появился, возможно, еще незрелый, но очень серьезный конкурент.

В качестве обоснования альтернативных подходов традиционно указываются слабые места стандарта: узкая ориентация языков, их взаимозависимость, низкие структурирующие возможности и отсутствие единой методики описания управляющего алгоритма.

Также известны случаи успешных попыток модификаций и языков стандарта *МЭК 61131-3* производителями ПО, например, предпринятые компанией *S3* с целью вооружить пользователей пакета *CoDeSys* средствами объектно-ориентированного программирования. Такие факты также можно расценивать как признак определенной неудовлетворенности стандартом со стороны пользователей, а равно и того, что специалисты-разработчики ПО не только ищут, но и видят реальные способы модификации языков программирования *МЭК 61131-3*, позволяющие расширить круг потенциальных клиентов.

В статье обсуждаются возможные пути расширения языка *ST*, которые могли бы обеспечить возможность его самодостаточного использования для широкого спектра задач промышленной автоматизации. Для этого приводится специфика задач промышленной автоматизации, вводятся базовые понятия процесса и функции-состояния, а также предлагаются операнды, обеспечивающие совместное функционирование и взаимодействие процессов, описанных на языке *ST*.

Специфика задач автоматизации

Специфика автоматизации предполагает наличие собственно системы управления, включающей датчики обратной связи и органы управления, и внешней (по отношению к системе управления) среды, на которую система управления воздействует через органы управления, – *объекта управления* – технической системы, реализующей некоторую производственную технологию. Воздействия или, другими словами, *реакция* системы управления определяются алгоритмом управления в зависимости от *событий* на объекте управления, информация о которых поступает через датчики обратной связи. Для цифровых систем это обстоятельство обусловливает *цикличность* управляющего алгоритма по схеме: считывание состояния входных сигналов через датчики (их обработка) и формирование выходных сигналов – выдача выходных сигналов на исполнительные органы. *Событийность* предполагает

* Информация для ссылок: Зюбин В. Е. Пути расширения языка ST из состава МЭК 61131-3 для задач промышленной автоматизации // Приборы и системы. 2009. №3. С. 16-19.

алгоритмические изменения программы и набора обрабатываемых ею входных/выходных сигналов в зависимости от происходящих на объекте событий.

Алгоритм управления предполагает *синхронизацию* своего исполнения с физическими процессами во внешней среде, что обусловливает необходимость развитой службы времени и активную работу с временными объектами: задержками, паузами, таймаутами.

Другая характерная особенность алгоритмов управления – *логический параллелизм*, отражающий существование множества параллельно протекающих процессов в объекте управления. Поскольку события, происходящие в различных компонентах системы, возникают независимо и в произвольной последовательности, то попытка задать реакцию системы единым блоком означает комбинаторный перебор большого числа вариантов и неоправданный рост сложности описания. Логический параллелизм предполагает наличие в алгоритме управления независимых или слабо зависимых частей – логически обособленных потоков управления.

Поскольку программы пишутся человеком и исключительно для человека, то в силу особенностей человеческой психики языки должны быть просты в изучении. Кроме того, языки должны предоставлять *механизмы структуризации* алгоритма (в нашем случае – языковые средства организации совместного функционирования логически параллельных частей) и *механизмы абстрагирования* (в нашем случае – понятийный переход от датчиков и исполнительных органов к целевому ТП). То есть программа должна быть организована в виде обозримых, информационно-изолированных компонентов, возможно иерархически вложенных друг в друга, и на некотором уровне иерархии программирование должно вестись в естественных терминах ТП.

На основании перечисленных обстоятельств формальный язык программирования, пригодный для широкого спектра управляющих алгоритмов (УА), и его понятийный аппарат с необходимостью должен отражать следующую специфику задач автоматизации [5]:

- наличие внешней по отношению к алгоритму среды – объекта управления (ОУ);
- цикличность функционирования УА;
- присущую задаче управления событийность;
- необходимость синхронизации функционирования УА с физическими процессами на ОУ (служба времени);
- параллелизм физических процессов на ОУ.

Базовый понятийный аппарат для описания УА

Предварительный анализ проблемы показывает, что в качестве основы модели УА может быть использована модель конечного автомата (МКА). Практика программирования управляющих алгоритмов, в частности, и в рамках *МЭК 61131-3*, также подтверждает теоретический вывод о пригодности МКА для задач управления [6]. Однако прямое использование МКА для моделирования сложных УА невозможно, поскольку, с одной стороны, ее привлекательные свойства (возможность отразить взаимодействие с ОУ, событийность и цикличность) явно не подчеркнуты, а с другой стороны, МКА не предполагает операций с временны́ми интервалами и параллелизм. Кроме того, МКА ориентирована на схемную реализацию, что приводит к ряду негативных последствий. В частности, МКА сложна для понимания современным выпускником школы или ВУЗа, так как ее понятийный аппарат не соответствует текущим тенденциям в образовании, ориентированном на информационные технологии и тотальную компьютеризацию.

В качестве формальной модели сложного алгоритма управления, не имеющего перечисленных недостатков, была предложена модель гиперпроцесса (или гиперавтомата). В рамках этой модели управляющий алгоритм организуется как множество взаимодействующих друг с другом процессов – расширенного варианта конечных автоматов. По сравнению с конечным автоматом процессы имеют дополнительные механизмы для организации совместного и параллельного исполнения, а также средства синхронизации своего исполнения – возможность манипуляции с временными интервалами, паузами, задержками, таймаутами.

Модель гиперпроцесса была использована при создании *Си*-подобного языка *Рефлекс*. В отличие от языка *Си*, где программы строятся как иерархия функций, базовое понятие языка *Рефлекс* – процесс. Программа на языке *Рефлекс* – это множество параллельно исполняемых процессов, которые могут запускать друг друга, останавливать и контролировать текущее состояние.

Опыт применения языка *Рефлекс* на широком спектре задач промышленной автоматизации и полученные результаты свидетельствуют о его высоких эксплуатационных характеристиках [7].

При использовании языка *Рефлекс* на практике было обнаружено:

- существенное разложение программы в виде параллельно исполняемых взаимодействующих процессов (в реальных задачах число таких процессов достигало тысячи);
- возможность создания программных иерархий, отражающих естественное представление технологов об автоматизируемых производственных технологиях;
- богатство возможных стратегий и приемов, не имеющих аналогов в других областях программирования.

Это позволяет говорить о новом стиле программирования, которое было названо “*процесс ориентированным программированием*”.

Расширение *ST*

“Революционное” внедрение языка *Рефлекс* в среду пользователей средств *МЭК 61131-3* сопряжено с проблемным моментом: существование языка *ST*, ориентированного на язык *Паскаль*. Отличия между *Си*-подобным и *Паскале*-подобным синтаксисами языков затрудняют переход с *ST* на *Рефлекс*, а внедрение в стандарт языка *Рефлекс* как шестого языка неприемлемо в силу изначальной перегруженности *МЭК 61131-3*. Однако это обстоятельство не исключает воз-

можность внедрения в среду языков *МЭК 61131-3* процесс ориентированного программирования.

В качестве возможного варианта решения проблемы можно предложить доработку синтаксиса *ST*, аналогичную проведенной при расширении языка *Си* до *Рефлекса*. Как показывает предварительный анализ, доработка синтаксиса в данном случае будет весьма несущественной, поскольку доработка не затрагивает уже существующий синтаксис и семантику описания арифметических выражений и программных компонентов. Затраты на реализацию модели гиперпроцесса также не представляют сложности и хорошо описаны [8]. В качестве преимуществ, получаемых в результате доработки, следует назвать появление в стандарте *МЭК 61131-3* мощного выразительного средства, которое, с одной стороны, уже хорошо знакомо пользователям, а, с другой стороны, сочетает в себе и гибкие выразительные средства процедурного языка и понятийный аппарат, ориентированный на задачи автоматизации широкого класса.

Приведем вариант расширения *ST*-синтаксиса в *Паскале*-подобном стиле (см. таблицу).

Заключение

В статье предложен вариант расширения исходной процедурной программной модели языка *ST* до модели гиперпроцесса, позволяющий создавать управляющие алгоритмы широкого класса в стиле "*процесс ориентированного программирования*". Расширение достигается за счет введения нового типа программных компонентов – "*процесса*". Вывод о высоких эксплуатационных свойствах сконструированного языка основан на успешном опыте использования *Си*-подобного языка *Рефлекс*, также базирующегося на программной модели гиперпроцесса и позволяющего использовать приемы "*процесс ориентированного программирования*". Новые выразительные средства не исключают использование *ST* в текущем (*МЭК 61131-3*) виде, просто осваиваются пользователями и по оценкам автора несложно реализуемы в рамках существующих трансляторов языка *ST*. В результате расширения в *ST* появляется возможность описывать параллелизм, событийный полиморфизм и синхронизм – свойства, необходимые для комфортного программирования управляющих алгоритмов.

Работа выполнена Институтом автоматики и электрометрии СО РАН совместно с Новосибирским государственным университетом.

Контактный телефон (383) 330-71-62.
E-mail: zyubin@iae.nsk.su

Список литературы

1. *Зимин А.М., Букетин Б.В., Почуев А.П.* и др. Учебная Интернет-лаборатория "Испытания материалов" // Информационные технологии. 2006. № 10.
2. *Бурмистров А.В.* Новые возможности ЭНТЕК и KLogic: новые платформы, АСУТП электрических сетей, конечные автоматы в Klogic // Тезисы докладов IV Междунар. науч.-практич. конф.-выставки "Промышленные контроллеры 2008: от А до Я". Москва, 14-17 октября, 2008.
3. *Бутырин П.А., Васьковская Т.А., Каратаева В.В., Материкин С.В.* Автоматизация физических исследований и эксперимента: компьютерные измерения и виртуальные приборы на основе LabVIEW 7. М.: ДМК Пресс, 2005.
4. *Gates B.*A Robot in Every Home // Scientific American. January, 2006.
5. *Зюбин В. Е.* Программирование ПЛК: стандарт МЭК 61131-3 и возможные альтернативы // Промышленные АСУ и контроллеры. 2005. № 11.
6. *Петров И.В.* Отладка прикладных ПЛК программ в CoDeSys (часть 2) // Промышленные АСУ и контроллеры. 2006. № 3.
7. *Зюбин В.Е.* "Си с процессами": язык программирования логических контроллеров // Мехатроника. 2006. № 12.
8. *Зюбин В.Е.* Программирование информационно-управляющих систем на основе конечных автоматов: Учеб.-метод. пособие. Новосибирск: Новосиб. гос. ун-т, 2006, http://reflex-language.narod.ru/index.html

Таблица. Предлагаемое расширение синтаксиса языка *ST*

Резервированное слово	Пояснения по использованию
PROC **END_PROC**	Описание тела процесса. Например: **PROC** <имя процесса> <тело процесса> **END_PROC** Или, в общем случае, резервированное слово **PROC** является указанием на то, что далее следует имя процесса, например, **START PROC** <имя процесса>; **STOP PROC** <имя процесса>;
STATE **END_STATE**	Описание функции-состояния процесса. Например: **STATE** <имя состояния> <тело состояния> **END_STATE** Или, в общем случае, резервированное слово **STATE** является указанием на то, что далее следует спецификация имени функции-состояния, например, **IN STATE** <имя состояния>;
START	Оператор перевода указанного процесса в выделенное состояние "начальное состояние" (запуск процесса) **START PROC** <имя процесса>; Оператор обеспечивает вывод процесса из пассивного состояния, порождая тем самым параллельный поток управления.

STOP	Оператор перевода указанного процесса в выделенное состояние “нормальный останов” (нормальный останов процесса) Например, **STOP PROC** <имя процесса>; Или спецификация функции-состояния “нормальный останов” при операциях контроля функции-состояния процесса **IF PROC** <имя процесса> **IN STATE STOP THEN** <действия> **END_IF**
ERROR	Оператор перевода указанного процесса в выделенное пассивное состояние “останов по ошибке” (ошибка процесса) Например, **ERROR PROC** <имя процесса>; Или спецификация функции-состояния “останов по ошибке” при операциях контроля функции-состояния процесса **IF PROC** <имя процесса> **IN STATE ERROR THEN** <действия> **END_IF** Возможность проверки пассивных функций-состояний процесса (пассивный функций-состояний **STOP** и **ERROR**) позволяет контролировать момент окончания исполнения процесса и, тем самым, естественным образом организовывать надежную конвергенцию потоков управления.
TIMEOUT **END_TIMEOUT**	Оператор контроля времени нахождения текущего процесса в текущей функции-состоянии **TIMEOUT** t#5d14h12m18s3.5ms <действия> **END_TIMEOUT** Оператор позволяет самым естественным образом оперировать с временными интервалами при контроле времени срабатывания исполнительных устройств, фиксации тайм-аутов при связи по сети, организации тактирования циклов ПИД регулирования.