ЯЗЫК «РЕФЛЕКС» – ДИАЛЕКТ СИ ДЛЯ ПРОГРАММИРУЕМЫХ ЛОГИЧЕСКИХ КОНТРОЛЛЕРОВ*

В.Е. Зюбин
Институт автоматики и электрометрии СО РАН,
рук. тематической группы «Языки технологического программирования»
E-mail: zyubin@iae.nsk.su

Рассматривается язык Рефлекс, называемый иногда «Си с процессами», предназначенный для описания алгоритмов функционирования программируемых логических контроллеров. Обсуждаются базовые свойства и принципы, заложенные при создании языка.

Введение

Более двадцати лет назад с появлением дешевых микроконтроллеров началось их успешное внедрение в качестве базового элемента систем управления, и в настоящее время автоматизация исследовательских и промышленных комплексов уже не представляется возможным без использования цифровой техники. Цифровые системы управления обеспечивают проведение высокоточных научных экспериментов в области физики, химии, биологии, при исследовании космоса, определяют уровни обороноспособности страны и ее промышленного развития.

Наиболее широкую известность в промышленности микроконтроллеры получили под видом программируемых логических контроллеров (ПЛК), обычно связанных воедино с базами данных и средствами мультимедийного интерфейса с оператором.

Использование ПЛК (как и любых других устройств на базе процессорных элементов) предполагает наличие языков их программирования – языков создания программного обеспечения (ПО), алгоритмов функционирования ПЛК. При этом, на фоне постоянного роста производительности микроконтроллеров и снижения цен на аппаратуру, затраты на создание ПО ПЛК остаются практически неизменными. Это объясняется, как постоянными затратами на сопровождение ПО (что связано с развитием аппаратных, архитектурных и технологических подходов), так и отсутствием на рынке программных средств, которые бы отвечали требованиям задач автоматизации. На практике наибольшее распространение получили языки стандарта МЭК-61131-3. Эти языки строятся на искусственных графических метафорах (реле – LD, микросхем – FBD), что обеспечивает их простое изучение низко квалифицированным персоналом, однако (по этой же причине) и крайне затрудняет их эффективное использование уже в задачах средней сложности.

В связи с этим представляет интерес поиск решения, которое, с одной стороны, обеспечивает легкость изучения и сокращение накладных расходов на сопровождение, а, с другой стороны, упрощает создание управляющих алгоритмов за счет использования адекватной метафоры.

В статье рассматривается специфика типовой задачи управления, особенности алгоритмов управления, формулируются требования к языку программирования ПЛК. Приводятся базовые концепции языка Рефлекс и обсуждаются результаты, полученные при практическом использовании языка в сложных проектах.

1. Характерные особенности цифровых управляющих алгоритмов

Специфика автоматизации предполагает наличие собственно системы управления, включающей датчики обратной связи и органы управления, и внешней (по отношению

* статья была представлена на шестой международной научно-практической конференции "Средства и системы автоматизации " CSAF'06 // Томск, 1-3 ноября 2005 г. Томск: ТУСУР, 2005

к системе управления) среды – *объекта управления* – технической системы, реализующей некоторую производственную технологию, на которую система управления воздействует через органы управления. Воздействия – или, другими словами, *реакция* системы управления, – определяется 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: считывание состояния входных сигналов – их обработка и формирование выходных сигналов – выдача выходных сигналов.

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 слабозависимых частей.

Независимость и неопределенность при возникновении событий на объекте управления в случае параллельных алгоритмов может привести к конфликтам между одновременно выполняемыми частями алгоритма, называемым «гонками». Факт гонок трактуется как некорректность алгоритма. Поэтому методика создания алгоритма управления должна обеспечивать отсутствие таких ситуаций. На практике это достигается либо через процедуру формальной верификации алгоритма, либо обеспечивается конструктивными особенностями средств разработки, гарантирующими принципиальную невозможность гонок.

2. Цели создания языка Рефлекс

При разработке языка Рефлекс преследовались следующие цели:

- Язык должен быть адекватен решаемым задачам при организации общей структуры сложного управляющего алгоритма (обеспечивать событийность, цикличность, синхронизацию с процессами на объекте и службу времени, логический параллелизм, конструктивно обеспечиваемую надежность создаваемых алгоритмов).
- Язык должен быть легок для изучения.
- Язык должен быть удобен в использовании (предоставлять единую методику создания программ, исключать рутинные операции связи с датчиками и исполнительными органами, скрывать внутреннюю организацию параллелизма, допускать русскоязычный синтаксис).
- Язык должен быть прост в сопровождении, в частности, при переносе на произвольную вычислительную платформу.
- Язык должен допускать вызов внешних функции, реализованных в рамках «классического» процедурного подхода.

Анализ представленных на рынке языков показывает, что ни один из них не удовлетворяет приведенным требованиям в достаточной степени. В качестве прототипа могут быть указаны: а) язык ST (совместно с SFC) из стандарта МЭК 61131-3 [1], и б) языки группы finite state machine (FSM), наиболее удачный представитель – QuickStep [2]. Решение ST (+SFC) критикуется главным образом за сложность организации логического параллелизма, отсутствие единой методики создания программ. Язык QuickStep не может быть использован в широкой практике из-за ограничений на степень логического параллелизма (всего 28 параллельных процессов) и непереносимость (жесткую привязку синтаксиса к аппаратуре фирмы Control

Technology Corporation). Таким образом, разработка нового языка вполне обоснована и правомерна.

3. Базовые концепции языка Рефлекс

В качестве математической модели программы на языке Рефлекс была предложена модель гипер-автомата [3]. Гипер-автомат задается совокупностью процессов, которые активизируются с заданной периодичностью. Процесс представлен набором своих состояний-функций и текущим состоянием. Состояние задается через события и реакцию на эти события. Активизация процесса заключается в выполнении его текущего состояния. Смена текущего состояния – одна из возможных реакций на события. Процесс строится как расширение классического конечного автомата (С-автомата). Такое расширение привело к существенной коррекции математической модели конечного автомата. Ревизия классической теории конечного автомата отразила объективную необходимость ввести в модель свойства цикличности, синхронности, событийности; предусмотреть совместное функционирование процессов, их взаимодействие; и адаптировать терминологию к современному уровню, достигнутому в программировании.

На уровне языка это выразилось в следующем:

- был введен новый концепт – «процесс», допускающий логическое распараллеливание потока управления, а с точки зрения теории программирования представляющий собой полиморфную функцию, реализующую событийный полиморфизм;
- мы отказались от алфавитной формы задания внешних связей и взамен перешли к переменным, что с одной стороны обеспечивает гибкость описания автоматов с большими входными/выходными алфавитами, а с другой – позволяет приблизить формат описания алгоритма к общепринятому в современном программировании уровню;
- был определен механизм взаимодействия между автоматами (решена проблема блочного синтеза). Введены операции «включения» автомата и его «выключения». Введены операции считывания текущего состояния автомата. Введены дополнительные (т.н. пассивные) состояния автомата при его останове – состояние «нормального останова» и состояние «останова по ошибке»;
- были введены операции с временны*ы*ми интервалами;
- специфицирован многопоточный логический параллелизм совместного функционирования автоматов с гранулярностью на уровне состояний.

Язык Рефлекс был выполнен как диалект языка Си, синтаксис которого допускает русскоязычный вариант. Было определено алгоритмически эквивалентное преобразование программы на языке Рефлекс в программу на языке Си, что кардинально упростило перенос языка на большинство существующих вычислительных платформ.

Таким образом, в результате удалось создать языковое средство, с одной стороны, достаточно мощное, по своим выразительным средствам и адекватное задаче программирования управляющих алгоритмов, а с другой стороны – в силу широкого распространения Си, простое для сопровождения и освоения, в частности, русскоязычным пользователем. Основные цели разработки языка Рефлекс были достигнуты.

4. Методика программирования на языке Рефлекс

Типовая методика проектирования программ на языке Рефлекс предполагает двухэтапный способ разработки алгоритмов, обоснованный в [4], который на начальном этапе проектирования алгоритма предполагает использование графических средств спецификации, а на этапе кодирования – текстового языка (собственно языка Рефлекс). На начальном этапе подход обеспечивает удовлетворение ряда психологических требований, связанных с нагрузкой на кратковременную память,

позволяет организовать конструктивный диалог с заказчиком, обеспечивает ракурсное рассмотрение проблемы и выявление наиболее подходящего способа алгоритмизации решения. На этапе кодирования – при формализованном представлении алгоритма в виде, пригодном для исполнения в цифровых системах, – подход предполагает использование текстовых языков, которые обеспечивают полноаспектную спецификацию, большую гибкость и модифицируемость создаваемых текстов, а также простоту сопровождения. Наличие бесшовного перехода от графической формы спецификации управляющих алгоритмов в виде логических блок-схем к языку Рефлекс показано в [5].

При создании языка учитывались психологические ограничения человека на обработку информации [4]. Для преодоления этих ограничений в языке предусмотрены следующие слои информационной изоляции, упрощающие пользователю создание, изучение и сопровождение алгоритма:

- При переходе от понятия модулей УСО к понятию порты ввода/вывода, что позволяет устранить из рассмотрения информацию об адресах УСО по системной магистрали и архитектурных особенностях УСО.
- При переходе от портов ввода/вывода к программным переменным, что обеспечивает локализацию информации о разводке сигналов и позволяет сформировать семантически «чистые» идентификаторы переменных.
- При переходе от переменных разного типа к понятию событий и реакции на события, скомпонованных в состояния, что позволяет представлять алгоритм как череду сменяющихся функций, ситуативно-адекватных течению технологического процесса. Концепт «состояние» обеспечивает разгрузку кратковременной памяти пользователя (решение проблемы закрытия [4]) за счет того, что анализ семантики текущего состояния предполагает рассмотрение только возможных выходов из состояния. Исторические причины, приведшие в это состояние, не важны, т.е. не требуют рассмотрения при программировании.
- При компоновке состояний в процессы, при переходе к целевым функциям, отвязанным от исполнительных органов, что придает локальных характер возможным модификациям алгоритма.
- При организации алгоритма в виде гибкой иерархической структуры процессов, семантически и терминологически адекватной предметной области автоматизируемого объекта.
- При трансляции текста программы в исполняемые коды (за счет автоматизации рутинных операций по привязке переменных к интерфейсной аппаратуре и реализации логического параллелизма исполнения взаимодействующих процессов, которые не нужно рассматривать при работе с текстом программы).

Такие свойства языка позволяют проектировать алгоритм в терминологии технологического процесса по методике «сверху-вниз», естественной при разработке конструкторской документации на технологический процесс: например, в терминах «вакуумирование», «контроль исходного положения», «включение регулятора температуры» и т.п. Привязка к конкретным исполнительным органам производится на заключительных этапах, однако, и при этом также может сохраняться специфика автоматизируемого объекта. Например, вместо «открытие клапана» – «подача аргона», вместо «выключение насоса» – «прекращение вакуумирования» и т.п. Привязка к интерфейсной аппаратуре задается независимым описанием связей используемых переменных и модулей входов/выходов.

Создание программы на языке «Рефлекс», по сути, просто отражает алгоритм, с необходимостью уже разработанный при создании объекта управления. Этим обуславливается субоптимальная структуризация алгоритма, легкость его изучения и сопровождения.

Отладка алгоритма производится имитационным моделированием, либо программно. Исполнение контролируется системными функциями мониторинга.

5. Отработка языка Рефлекс на реальных задачах

Наиболее масштабными проектами, выполненными на основе языка «Рефлекс», была серия систем управления для установок выращивания монокристаллического кремния методом вытягивания из расплава (метод Чохральского). Автоматизируемый технологический процесс предполагает плавление 90 кг агрессивного металла при температуре 1400 градусов Цельсия и затем контролируемую кристаллизацию его в виде полутораметровой «були» диаметром до 250 мм [6].

Объект управления характеризуется высокой сложностью и неоднородностью, что позволяет провести серьезную натурную проверку технологии разработки. Распределенная система управления предполагает: цифровые датчики, запорно-регулирующие и отсечные клапаны, вакуумные насосы, регулирование, четыре координаты (шесть приводов), источник питания 180 кВт, комплекс жестких требований по надежности и устойчивости. Специфика автоматизации: капризный технологический процесс (фазовый переход расплав-кристалл), полная автоматизация и ведение процесса без участия оператора, большой объемом экспериментальных работ при отработке, отсутствие право на ошибку (нештатное развитие ситуации приводит к частичному разрушению установки).

При выполнении проекта были получены следующие интересные результаты:

- обнаружена высокая степень разложения алгоритма управления на процессы: в целом алгоритм представлен 760-ю (sic!) параллельными процессами;
- подтверждена простота переноса языка на новую платформу: проведена за неделю;
- зафиксированы низкие накладные расходы на обслуживание процессов – порядка двух микросекунд на процесс (платформа MicroPC, CPU686E [7]);
- в режиме пиковой нагрузки теоретически достигаемое время реакции для произвольного события системы равняется восьми миллисекундам (в проекте использовалось равное 100 мс);
- при работе с языком действительно не требуется высокая квалификация: в разработке программ участвовали исполнители, не имеющие программистского образования;

В целом характеристики языка соответствовали теоретическим ожиданиям:

- в процессе разработки алгоритма активно участвовали заказчики-конструкторы, как на начальных стадиях, так и на этапе экспериментальной отработки;
- поскольку проект имел большую экспериментальную составляющую, алгоритм часто модифицировался, модификации затрагивали разные части алгоритма, однако принятый метод структуризации обусловил локальный характер проводимых изменений текста программы;
- язык обеспечил разработку алгоритма управления в терминах технологического процесса: семантическая бесшовность уровней технологии и языка ее описания привела к существенному сокращению объема формальных тестов – прозрачность алгоритма практически исключает возникновение логических ошибок;
- решения, заложенные при кодогенерации и реализации системных функций, конструктивно обеспечили устойчивость алгоритма: арифметические ошибки типа деления на ноль, плавающей арифметики и т.п. приводят лишь к локальным отказам параллельных частей алгоритма;
- от пользователей языка Рефлекс, были получены положительные отзывы; и, что особенно ценно, программисты, имеющие опыт работы с языками МЭК 61131-3 (продукт ISaGRAF), характеризовали язык Рефлекс как более гибкий и удобный по сравнению со средствами МЭК 61131-3.

Полученные при отработке языка Рефлекс результаты убедительно продемонстрировали его практическую пользу и эффективность.

Заключение

Учет специфики алгоритмов управления и психологических аспектов программирования позволил создать диалект Си, ориентированный на задачи автоматизации. Язык был назван «Рефлекс» или «Си с процессами», т.к. базовая семантика Си была расширена концептом «процесс», обладающем свойством цикличности исполнения и параллельности. Основными целями разработки языка «Рефлекс» были удобство описания управляющих алгоритмов, легкость изучения и сопровождения. Синтаксис языка имеет средства описания портов УСО, рутинные операции по преобразованию входных/выходных портов во внутренние программные переменные автоматизированы. Допускается использование русскоязычных идентификаторов и русскоязычного синтаксиса, что делает язык особенно привлекательным для отечественных пользователей.

Язык прошел проверку в реальных проектах. Проекты совмещали сложную логику управления и широкий спектр характерных задач промышленной автоматизации. Язык показал высокую надежность и удобство.

ЛИТЕРАТУРА

1. IEC 65B/373/CD, Committee Draft - IEC 61131-3. Programmable controllers. Part 3: Programming languages, 2nd Ed. // International Electrotechnic Commission. – 1998.
2. Control Technology Corporation. Doc. No. MAN-1010A. Quickstep™ Language and Programming Guide // Control Technology Corporation. – 2000. [http://www.ctc-control.com/customer/techinfo/docs/QuickstepLangProg.pdf]
3. Зюбин В.Е, Петухов А.Д. Распределение вычислительных ресурсов в средах с многопоточной реализацией гипер-автомата // Труды III Международной конференции «Идентификация систем и задачи управления» SICPRO ’04 Москва 28-30 января 2004 г. – С. 446-465.
4. Зюбин В.Е. Графика и текст: какой язык выбрать программисту // Открытые системы, 2004, №1. – С. 54-58.
5. Зюбин В.Е. Графические и текстовые формы спецификации сложных управляющих алгоритмов: непримиримая оппозиция или кооперация? // Сб. Тр. VIII Международная конференция по электронным публикациям "EL-Pub 2003" (8-10 октября 2003 г., г. Новосибирск, Академгородок) [http://www-sbras.nsc.ru/ws/elpub2003/]
6. Зюбин В.Е., Котов В.Н., Котов Н.В. и др. Базовый модуль, управляющий установкой для выращивания монокристаллов кремния // Датчики и системы, 2004, № 12. – С. 17-22.
7. CPU686E. Модуль процессора. Руководство пользователя. Fastwel Inc., 1999.