В.Е. ЗЮБИН (Институт автоматики и электрометрии СО РАН)

Программирование ПЛК: языки МЭК 61131-3 и возможные альтернативы

Статья посвящена языкам для программируемых логических контроллеров. Рассматриваются языки международного стандарта МЭК 61131-3, обсуждаются проблемы стандарта и возможные альтернативы.

Статья опубликована в журнале "Промышленные АСУ и контроллеры". – №11. – 2005.– С.31-35

В 1993 г. Международная электротехническая комиссия выпустила в свет стандарт МЭК 61131-3. Этот международный стандарт входит в группу МЭК 61131 стандартов, которые охватывают различные аспекты использования ПЛК. Декларируемые цели МЭК 61131-3 – стандартизация существующих языков ПЛК [1], а вернее, базовая платформа для такой работы в национальных комитетах стандартизации. Унификация сотни существующих версий языков – дело благородное, и такие усилия можно было бы однозначно приветствовать, если бы не некоторые обстоятельства, сопутствующие этому процессу.

В статье изложены предпосылки создания МЭК 61131-3, структура организаций, связанных со стандартом, и их функции, недостатки, заложенные при разработке стандарта, а также прагматические аспекты использования языков стандарта, которые следует учитывать при выборе средств программирования ПЛК. В конце статьи кратко обсуждаются возможные альтернативы.

ПЛК как ядро систем автоматики. Специфика задачи и предпосылки создания специализированных языков

На современном этапе в качестве ядра любой системы промышленной автоматизации используется ПЛК, к которому со стороны объекта автоматизации подключаются датчики и исполнительные органы. Через датчики в ПЛК поступает информация о текущем состоянии объекта, а через исполнительные органы ПЛК может изменять состояние управляемого объекта. Эта базовая схема может усложняться. Например, ПЛК могут подключаться к АРМ оператора для супервизорного управления или к БД для накопления информации и интеграции в АСУ предприятия. Поскольку все ПЛК строятся на базе цифровой техники, естественным образом предполагаются некоторые языковые средства их программирования. Причем в силу специфики задачи алгоритмические языки программирования, такие как Си, Паскаль, Си++, не годятся для этих целей.

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 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 технологическому процессу). Т.е. 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 технологического процесса.

Перечисленные обстоятельства обусловливают разработку специализированных языков промышленной автоматизации.

Международная электротехническая комиссия. Цели создания стандарта на языки программирования ПЛК

Международная электротехническая комиссия – это международный орган стандартизации, создающий базовые стандарты для последующей адаптации

в национальных комитетах. Интересный факт, которым могут гордиться граждане России. В становлении и работе этой комиссии принимал активное участие СССР, поэтому русский – это один из трех официальных языков МЭК. Что касается стандартизации языков, используемых для программирования ПЛК, то эта проблема назрела давно. К концу 80-х десяток базовых концепций на практике был представлен более сотней вариаций. Их унификация сулила ощутимый экономический эффект. Для решения этой проблемы была создана рабочая группа, состоящая из представителей ведущих игроков на рынке автоматизации, которая начала работу.

В силу того, что общепринятого подхода к программированию ПЛК не существовало (и не существует до сих пор), членам комиссии не удалось договориться о едином языке. Поэтому было принято компромиссное решение – включить в стандарт языки, используемые в фирмах, представителям которых посчастливилось оказаться в членах группы.

Среди языков-“счастливцев” оказались:

- *SFC* (Sequential Function Chart) – графический язык, используемый для описания алгоритма в виде набора связанных пар: шаг (step) и переход (transition). Шаг представляет собой набор операций над переменными. Переход – набор логических условных выражений, определяющий передачу управления к следующей паре шаг-переход. По внешнему виду описание на языке SFC напоминает хорошо известные логические блок-схемы алгоритмов, хотя идеологически SFC близок к сетям Петри. SFC имеет возможность распараллеливания алгоритма. Однако SFC не имеет средств для описания шагов и переходов, которые могут быть выражены только средствами других языков стандарта. Происхождение: Grafcet (Telemechanique-Groupe Schneider).
- *LD* (Ladder Diagram) – графический язык, стандартизованный вариант класса языков релейно-контактных схем. Логические выражения на этом языке описываются в виде реле, которые широко применялись в области автоматизации в 60-х годах. Из-за своих ограниченных возможностей язык дополнен привнесенными средствами: таймерами, счетчиками и т.п. Происхождение: различные варианты языка релейно-контактных схем (Allen-Bradley, AEG Schneider Automation, GE-Fanuc, Siemens).
- *FBD* (Functional Block Diagram) – графический язык, по своей сути похожий на LD: вместо реле в этом языке используются функциональные блоки. Алгоритм работы некоторого устройства, выраженный средствами этого языка, напоминает функциональную схему электронного устройства: элементы типа “логическое И”, “логическое ИЛИ” и т.п., соединенные линиями. Корни языка выяснить сложно, однако большинство специалистов сходятся во мнении, что это ни что иное, как перенос идей языка релейно-контактных схем на другую элементную базу.
- *ST* (Structured Text) – текстовый высокоуровневый язык общего назначения, по синтаксису ориентированный на Паскаль. Самостоятельного значения не имеет: используется только совместно с SFC. Происхождение: Grafcet (Telemechanique-Groupe Schneider).
- *IL* (Instruction List) – текстовый язык низкого уровня. Выглядит как язык ассемблера, что объясняется его происхождением: для некоторых моделей ПЛК фирмы Siemens является языком ассемблера. В рамках стандарта IEC 1131-3 к архитектуре конкретного процессора не привязан. Самостоятельного значения не имеет: используется только совместно с SFC. Происхождение – STEP 5 (Siemens).

Основные недостатки МЭК 61131-3. Мифологемы стандарта

Даже при первом взгляде виден перекос в выборе языков. *С одной стороны*, в стандарт попадает ассемблер (IL), *с другой стороны*, чрезвычайно мощная ветка т.н. языков машин конечных состояний FSM оказывается за рамками рассмотрения. *Другим*, некорректным и дезориентирующим пользователей, решением МЭК стала группировка языков в едином стандарте, что в корне отличается от общепринятых подходов к стандартизации языков программирования, см., например, ANSI C, Ada и т.д. *Третьим*, технически слабым, местом стандарта стало исключение вопросов унифицированного представления графических языков стандарта, что автоматически обусловило проблемы совместимости продуктов разных производителей. Кроме этого, в стандарте МЭК61131-3 не рассматривается вопрос привязки алгоритма к интерфейсной аппаратуре, которая с необходимостью присутствует в любой системе управления.

Вокруг стандарта МЭК 61131-3 возникло несколько мифологем, основные из которых рассмотрены ниже.

Мифологема 1. Для программирования ПЛК допускается применять только языки стандарта МЭК 61131-3.

Комментарий. Это неверно. Для программирования ПЛК могут применяться произвольные языки и, естественно, наиболее адекватные решаемой задаче. Более того, во многих (если не в большинстве) средств программирования, ориентированных на МЭК 61131-3, язык Си – это, по сути, шестой язык программирования. Многие ПЛК по-прежнему программируются и языками класса FSM, и на языках Си/Си++.

Мифологема 2. Среда разработки МЭК 61131-3 должна обеспечивать возможность программирования на всех пяти языках.

Комментарий. Это неверно. Языки стандарта независимы, и разработчиками стандарта предполагалось, что средство программирования МЭК 61131-3 поддерживает только один язык из набора. Более того, мультиязыковые программы расцениваются как потенциально ненадежные и затратны в сопровождении. На практике нарушение этого принципа языковой однородности программ всегда вынуждено и просто отражает ограничения, присущие языкам МЭК 61131-3.

Мифологема 3. МЭК 61131-3 обеспечивает кросс-брендовую и кросс-платформенную переносимость пользовательских программ.

Комментарий. Это неверно. Стандарт специфицирует лишь внешний вид, синтаксис языков, что для графических языков (в отличие от текстовых) недостаточно для переносимости. Выбрав фирму-поставщика, пользователь с большой вероятностью навсегда связывает с ней свою судьбу. Также стандарт допускает создание несовместимых вариаций и регламентирует лишь наличие списка несоответствий стандарту.

Если посмотреть на ситуацию непредвзято, можно заметить, что перечисленные недостатки и дезориентирующие мифологемы на руку мега-корпорациям, которые пытаются защитить себя и свой сегмент рынка от нежелательной конкуренции. Не останавливаясь подробно на этом аспекте, можно посоветовать читателю хорошую статью об использовании стандартизации в конкурентной борьбе [2]. Существуют и другие более жесткие мнения, в которых стандарт МЭК 61131-3 прямо называется контр-продуктивным [3].

PLCopen. Попытка адаптировать стандарт для целей сообщества конечных пользователей

Обеспечение кросс-брендовой переносимости и реальная стандартизация языков программирования ПЛК весьма актуальны для индустрии. Функции сертификации и развития стандарта взяла на себя независимая организация PLCopen [4], которая попыталась адаптировать стандарт МЭК 61131-3 к нуждам конечных пользователей.

PLCopen объединила средних и мелких игроков на рынке ПЛК, системных интеграторов и пользователей. Внутри PLCopen долгое время действовала группа по разработке независимого формата, обеспечивающего кросс-брендовую переносимость пользовательских программ. Была развернута активность по формированию и продвижению корректировок стандарта, отвечающих нуждам сообщества. Была разработана процедура и организована работа экспертных комиссий по сертификации CASE-средств, ориентированных на стандарт.

К сожалению, базовые подходы, использованные при разработке стандарта, оказали серьезное сопротивление деятельности PLCopen. Как следствие, работы по созданию переносимых программ были практически прекращены, FxF (File-eXchange-Format) не получил поддержки. Наиболее значимым результатом по корректировке стандарта явился, пожалуй, ввод локальных переменных. Сертификация CASE-средств ограничивается языками ST и IL (читай, клонами Паскаля и ассемблера – текстовыми языками, не имеющими самостоятельного значения). В силу этого обстоятельства проблема переносимости программ МЭК 61131-3 остается по-прежнему нерешенной. К большому сожалению, скептические прогнозы, сделанные в [5] относительно будущего стандарта МЭК 61131-3, стали реальностью.

Прагматика языков МЭК 61131-3. Выбор языка под конкретную задачу

Несмотря на недостатки, существуют вполне определенные ситуации, когда языки МЭК 61131-3 могут быть использованы на практике. В относительно простых задачах, не предъявляющих строгих требований по надежности, языки МЭК могут оказаться экономически эффективными.

Даже в ситуации, когда языки МЭК слабо подходят для практической задачи, отказ от их использования совсем не очевиден. В первую очередь, это вызвано тем обстоятельством, что конечному пользователю или системному интегратору тяжело конкурировать с мега-корпорациями, разрабатывать и поддерживать альтернативные решения. Ведь, кроме собственно языка, в среду разработки входит набор вспомогательных программ и библиотек, существенно облегчающих работу по тестированию и настройке системы. Поэтому, несмотря на недостатки, языки МЭК 61131-3 вполне допустимо использовать. Единственно, не следует выбирать язык реализации вашего проекта случайным образом. Базовые знания о психологии программирования и анализируемых свойствах языков, необходимые для проведения такой работы, изложены в [6].

Ниже рассмотрены характеристики языков МЭК 61131-3 с точки зрения прагматики их использования и даны некоторые рекомендации по выбору.

1. *LD* – метафора реле, лежащая в основе концепции, графическая форма описания алгоритма, позволяет легко освоить язык непрофессионалу. При переходе на ПЛК язык обладал вполне объяснимыми преимуществами, т.к. снимал психологические проблемы переучивания персонала. Отладка логической программы производится естественным образом с использованием исходной записи. Как и любой метафорический язык, LD не обладает должной степенью универсальности, ориентирован на манипуляции с битовыми переменными, имеет ограничения на сложность описываемого алгоритма. Цикличность и логический параллелизм на уровне инструкций – неотъемлемые атрибуты LD.

Синхронизация, операции с аналоговыми сигналами, структуризация алгоритма вызывают большие проблемы. Абстрагирование и отход от ключевой метафоры реле невозможны. Событийность алгоритма может быть обеспечена только инородными для подхода средствами и чрезвычайно сложна в реализации.

Типовой пример использования – реализация аварийных блокировок системы, включающая преимущественно логические сигналы. Вполне приемлем в случаях, когда задачей предполагаются частные коррекции несложного логического алгоритма неквалифицированным (с точки зрения программирования) персоналом (ремонтники, механики и т.п.).

2. *FBD* – обладает характерным для метафорических языков преимуществом: легкостью начального изучения. Предоставляет достаточно естественную возможность работы с аналоговыми переменными и минимальные средства структуризации (новые функ-

циональные блоки можно компоновать, используя уже существующие). Языку присущ логический параллелизм. Средства синхронизации достаточно естественны для языка.

Существенно меньшая по сравнению с LD наглядность при отладке программы. Проблемы с обработкой логических конструкций: средства управления потоком операций – конструкции типа if-then-else – отсутствуют среди выразительных средств языка, что ограничивает универсальность языка и часто приводит к появлению нестандартных библиотечных блоков, создаваемых сторонними средствами, например, на языке Си. Сильная степень структуризации алгоритма связана с ростом входных/выходных переменных функциональных блоков, что ограничивает сложность описываемых алгоритмов. Соответственно, ограничены и абстрагирующие свойства языка: информация о физических сигналах связи с объектом управления практически неустранима. Подход характеризуется отсутствием событийности.

Типовой пример использования – алгоритмы регулирования, обработка (например, фильтрация) аналоговых сигналов. В качестве пользователей предполагаются специалисты в области автоматического регулирования с привлечением квалифицированных системных программистов в сложных случаях.

3. *SFC+ST* – это, пожалуй, наиболее мощное и универсальное средство из состава языков МЭК 61131-3. Графика языка SFC облегчает изучение языка. Наличие общих корней с сетями Петри частично снимает проблемы синхронизации и параллелизма. Программирование операций с аналоговыми и логическими переменными достаточно комфортно за счет использования текстового Паскаль-подобного языка ST. Управление потоком команд не вызывает проблем. Существенное преимущество подхода – событийность, естественным образом поддерживаемая через механизм “шаг-переход”. Отладка программ может быть облегчена визуальной трассировкой потока управления. Слабые места языка SFC (как и сетей Петри) – это абстрагирование и структурирование [7], что, как и в предыдущих случаях, негативно сказывается на сложности программируемых алгоритмов и их качестве (сопровождаемости и надежности). По сравнению с LD и FBD подход SFC+ST проигрывает в удобстве программирования параллелизма алгоритма и, соответственно, более подходит для программирования линейных алгоритмических последовательностей.

Типовой пример использования – совмещенные алгоритмы логического и аналогового управления. В качестве пользователей предполагаются программирующие специалисты, совмещающие знание алгоритмических языков программирования и специфики автоматизируемого технологического процесса.

4. *SFC+IL* – по-видимому, не стоит рекомендовать к использованию вообще. В качестве примера применения такого экзотического варианта можно предложить гипотетический случай, когда в распоряжении имеется единственный программирующий специалист, знакомый только с ассемблером линейки ПЛК производства Siemens.

Следует добавить, что использование языков МЭК 61131-3 может обеспечить упрощение программирования и системную интеграцию, т.к. для имеющихся на рынке МЭК-средств, как правило, существуют более-менее апробированные решения, которые можно использовать в качестве прототипов вашей системы. С другой стороны, решение об использовании МЭК-средства имеет смысл предварять тщательным анализом требований задачи, в случае, если вы столкнетесь с ограничениями, убедить производителя учесть в среде разработки специфику вашего проекта будет весьма непросто. В качестве начального пособия по изучению МЭК-средств настоятельно рекомендуется книга И.В. Петрова [8].

Возможные альтернативы

В качестве альтернативы можно попытаться использовать средства, встроенные в SCADA-системы. Разумеется, такое решение допустимо лишь в некритичных задачах супервизорного контроля.

В большинстве случаев штатная эксплуатация систем управления не предусматривает наличие средств проектирования алгоритмов. Стандартный подход – это создание базового алгоритма с возможностью его настройки через ограниченное число доступных пользователю параметров. Это позволяет проектировать базовый алгоритм управления квалифицированными специалистами и адекватными языковыми средствами, а сопутствующую этому сложность “скрывать” за дружественным интерфейсом оператора, который создается, например, с помощью тех же SCADA-пакетов.

В принципе, допустимо решение об использовании для задач управления языков Си/Си++. Такой подход может быть оправдан при наличии штата квалифицированных специалистов, отлаженной культуре разработки ПО и больших объемах тиражируемых изделий. Си++ предоставляет хорошие возможности для адаптации языка к широкому спектру задач, так что создание паттернов и набора классов, ориентированных на приведенную выше специфику, вполне осуществимо. Однако при использовании алгоритмического языка для задач автоматизации невозможно обеспечить должный уровень контроля корректности программ, ее семантическую целостность. Сложность подхода – высокие квалификационные требования к программистам, существенные затраты на обеспечение надежности и низкая сопровождаемость программ, трудности с вовлечением в процесс разработки конечного пользователя.

Достаточно популярно в России обсуждение т.н. switch-технологии [9]. В основе подхода лежит известная реализация конечного автомата, в котором состояния автомата (некий набор функций) пронумерованы, а номер текущего состояния хранится в выделенной ячейке памяти. Текущая функция определяется через Си-конструкцию табличного выбора switch (этот факт и был использован при выборе названия). Дополнительно к этой базе предлагается на-

бор приемов по разработке алгоритма, его отладке, специфицируется идентификационная система для переменных. Подход обеспечивает цикличность, логический параллелизм, достаточную свободу в организации вычислений и, несомненно, имеет право на рассмотрение как вариант “пишу на Си”. Switch-подход успешно используется в учебном процессе. К сожалению, подход нуждается в проработке методов синхронизации, структуризации и абстрагирования. Не прописаны процедуры связи с УСО. Вызывает вопросы предлагаемая идентификационная система, присутствуют отклонения от действующих в России стандартов.

Как дополнительные варианты Си-подхода можно рассматривать работы [10-14], исследующие проблемы адаптации алгоритмических языков Си, Си++, Java к задачам автоматизации через создание специализированных библиотек, классов, паттернов, направленных на введение в базовые языки необходимых свойств параллелизма, событийности и структурности. Основной недостаток перечисленных подходов заключается в невозможности достигнуть необходимую степень комфортности программирования и безопасности получаемых программ. Остается очень большое число рутинных операций, выполняемых вручную. Контроль специфической семантики программ, появляющейся при расширении языков общего назначения, не автоматизирован. Отсутствует унификация привязки к УСО. Присутствует и следующий нюанс. Концептуально большинство таких подходов базируется на теории конечных автоматов, разрабатываемой, в первую очередь, для синтеза управляющих контроллеров из электронных компонентов. Это обстоятельство повлияло на понятийный аппарат теории и на современном этапе приводит к серьезным психологическим проблемам при освоении методики выпускниками учебных заведений, т.к. в настоящее время образование ориентировано на информационные технологи и программирование, а не на схемотехнику.

Особый интерес вызывает работа над стандартом МЭК-61499 [15], в котором разработчики предприняли попытку преодолеть ограничения языков МЭК61131-3 и скомбинировать в одном языковом средстве и поддержку логического параллелизма, и поддержку событийности. Цель стандарта – предоставить методологию разработки сложных алгоритмов. Программные компоненты представлены функциональными блоками специального вида: кроме обычных для языка FBD входных и выходных данных, интерфейс функционального блока стандарта МЭК-61499 предполагает событийные входы/выходы. Несомненно, это нововведение частично решает проблему событийности для классических функциональных блоков. К сожалению, этот несомненно прогрессивный стандарт не поддержан ведущими производителями ПЛК, и известные в настоящий момент реализации стандарта носят скорее исследовательский характер.

Для преодоления ограничений языков МЭК61131-3 на сложность алгоритма и недостатков прямого Си-подхода в Институте автоматики и электрометрии СО РАН был разработан специализированный язык программирования Reflex, который также может рассматриваться в качестве варианта. По своим свойствам язык вполне конкурентоспособен. При его разработке ставилась цель легкости освоения, сопровождения и соответствия задачам автоматизации.

Язык Reflex выполнен как диалект Си, что обеспечивает его легкое освоение. В проекте Reflex язык Си расширен понятием процесса – циклически исполняемой, параллельной сущности. Программа описывается как совокупность взаимодействующих процессов. Процессы можно запускать, останавливать, проверять их текущее состояние. Поэтому язык Reflex иногда называют “Си с процессами”. Предусмотрена гибкая структуризация алгоритма управления. Событийность алгоритма обеспечивается через механизм состояний процесса. Язык позволяет абстрагироваться от УСО и описывать алгоритм в терминах технологического процесса. Синтаксис Си расширен средствами синхронизации. При программировании на языке Reflex пользователь освобожден от рутинных действий и может полностью сконцентрировать свое внимание на сути создаваемого алгоритма. Математическая модель программы терминологически ориентирована на современные тенденции в образовании, что позволяет снять психологические проблемы при освоении методики [16].

Транслятор обеспечивает расширенную диагностику ошибок и контроль семантической корректности программы. Выходные файлы трансляции – файлы на языке Си. Кодогенерация обеспечивается штатным транслятором целевой платформы, что существенным образом снижает расходы на поддержание средства разработки и мобильность пользовательских программ. Переносимость языка базируется на Си: все платформо-зависимые части сконцентрированы в открытую для редактирования библиотеку. Отладка программ ведется в текстовом режиме. Предоставлены базовые функции трассировки потока управления, которые открыты для развития пользователем.

Язык (используется с 1990 г.) прошел серьезную проверку в серии проектов. В частности, в задаче автоматизации выращивания монокристаллического кремния, предполагающей работу с типичными для промышленной автоматизации устройствами (сетевые интеллектуальные датчики, 4-координатную систему перемещений, приводы, дублированную газовакуумную станцию, систему охлаждения, термосистему, контроль и упреждение аварийных ситуаций, набор аналоговых и дискретных входов). Аппаратная платформа – MicroPC (Fastwel), укомплектованная серийно выпускаемыми компонентами фирм Fastwel, Octagon Systems, Grayhill [17]. В настоящее время рассматривается вопрос о передаче языка Reflex в свободный доступ.

Заключение

Использование одного из языков стандарта МЭК61131-3 в реальных проектах может быть вполн-

не успешным, но при условии тщательной проработки вопроса соответствия свойств выбранного языка и требований задачи. Существенное преимущество МЭК 61131-3 подхода – наличие на рынке развитых сред разработки. При определенных обстоятельствах вполне допустимо использовать альтернативные средства программирования ПЛК, либо на основе FSM, таких, как язык Reflex, либо при известной осторожности и наличии высококвалифицированных кадров на основе чистого Си/Си++.

Владимир Евгеньевич Зюбин – кандидат технических наук, руководитель тематической группы «Языки технологического программирования», Институт автоматики и электрометрии СО РАН..
Телефон: (383) 330-71-62.
E-mail: zyubin@iae.nsk.su

Список литературы

1. *IEC 65B/373/CD, Committee Draft* – IEC 61131-3. Programmable controllers. Part 3: Programming languages, 2nd Ed // International Electrotechnic Commission. 1998.
2. *Арнольд Д.* Смысл графических стандартов: коммерческая выгода и риск // Программирование. 1996. №6.
3. *Crater K.* "When Technology Standards Become Counterproductive". Control Technology Corporation. 1996.
4. *PLCopen* Устав Ассоциации. 1997. (http://www.PLCopen.org/artass.htm)
5. *Зюбин В.Е.* К пятилетию стандарта IEC 1131-3. Итоги и прогнозы // Приборы и системы управления. 1999. №1.
6. *Зюбин В.Е.* Графика или текст: какой язык нужен программисту? // Открытые системы. 2004. №1.
7. *Анисимов Н.А., Голенков Е.А., Харитонов Д.И.* Композиционный подход к разработке параллельных и распределенных систем на основе сетей Петри // Программирование. 2001. №6.
8. *Петров И.В.* Стандартные языки и приемы прикладного программирования // М.: СОЛОН-Пресс. 2004.
9. *Шалыто А.А., Туккель Н.И.* SWITCH-технология – автоматный подход к созданию программного обеспечения "реактивных" систем // Программирование. 2001. №5.
10. *Яков Розенберг* Динамическая модификация алгоритмов. 2003.
11. *Шопырин Д.Г., Шалыто А.А.* Объектно-ориентированный подход к автоматному программированию. 2003.
12. *Любченко В.* О бильярде с Microsoft Visual C++ 5.0 // Мир ПК. 1998. № 1.
13. *Paul J. Lucas, Fabio Riccardi* Concurrent Hierarchical State Machine // Web-портал. 2004.
14. *Шамгунов Н.Н., Корнеев Г.А., Шалыто А.А.* State Machine – расширение языка Java для эффективной реализации автоматов // Санкт-петербургский государственный университет информационных технологий, механики и оптики. Кафедра "Технологии программирования". 2004.
15. *Christensen J.* IEC 61499 (Function Blocks for industrial-process measurement and control systems) FAQ. 1999.
16. *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. 2004.
17. *Зюбин В.Е., Котов В.Н., Котов Н.В. и др.* Базовый модуль, управляющий установкой для выращивания монокристаллов кремния // Датчики и системы. 2004. № 12.