Концепция числового программного управления мехатронными системами: проблемы управления электроавтоматикой

д.т.н., профессор
СОСОНКИН В.Л.

д.т.н. МАРТИНОВ Г.М.
МГТУ «СТАНКИН», Москва

Рассмотрены варианты управления электроавтоматикой мехатронных систем с помощью программируемых контроллеров.
Отмечена все возрастающая привлекательность программно-реализованных (виртуальных) контроллеров типа SoftPLC.
Рассмотрена общая организация управления типа SoftPLC: система понятий в соответствии со стандартом IEC 6133-3; альтернативные структуры клиентской части проекта системы управления; работа серверной части программы управления; объектный подход при управлении электроавтоматикой; особенности управления электроавтоматикой станков с ЧПУ. Отмечены две особенности управления электроавтоматикой станков с ЧПУ: задачи SoftPLC квазипараллельны задачам ЧПУ и работают в одной и той же исполнительной среде; циклы управления инициируются управляющей программой ЧПУ.

Классификация систем управления электроавтоматикой. На рис. 1 показаны варианты программируемых контроллеров. Тип 1 представляет собой традиционное решение, в то время как типы 2...4 с различной степенью глубины используют персональный компьютер. Так, для типа 2 персональный компьютер служит только средством организации интерфейса пользователя. В типе 3 привлекается дополнительный процессор для выполнения программы управления электроавтоматикой. Тип 4 использует вычислительную мощность персонального компьютера как для построения интерфейса пользователя, так и для выполнения всех управляющих функций. Этот тип относится к наиболее современному, перспективному и наиболее гибкому решению, которое получило наименование «программно-реализованный (виртуальный) контроллер, SoftPLC». Контроллерные функции здесь реализованы в виде приложения в персональном компьютере. Это позволяет строить как интерпретируемые, так и компилируемые системы управления электроавтоматикой. Нетрудно заметить, что представленные варианты различаются способом реализации и местом размещения клиентской (интерфейсной) и серверной (исполнительной) частей общей системы управления.

Система понятий, используемых при организации системы управления. Терминология стандарта IEC61131-3 в части организации системы управления ориентирована скорее на традиционные контроллеры. Программная система, удовлетворяющая IEC61131-3 проекту, содержит следующие компоненты: конфигурацию, ресурсы, задачи.

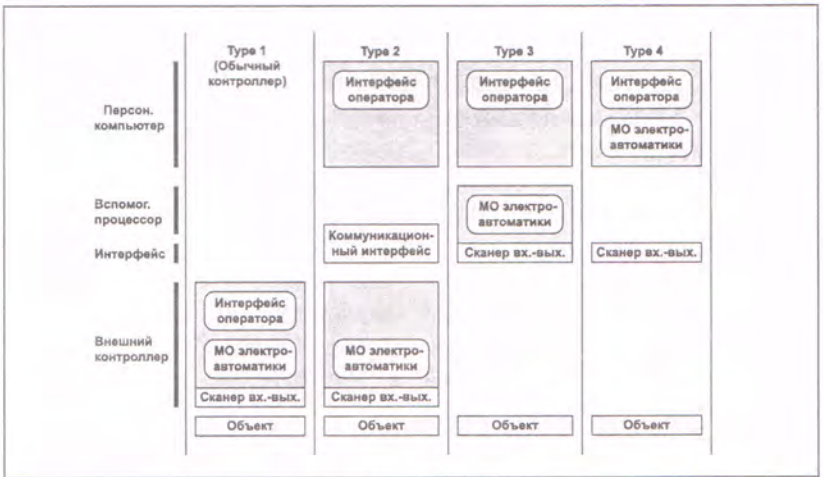

Рис. 1. Классификация программируемых контроллеров

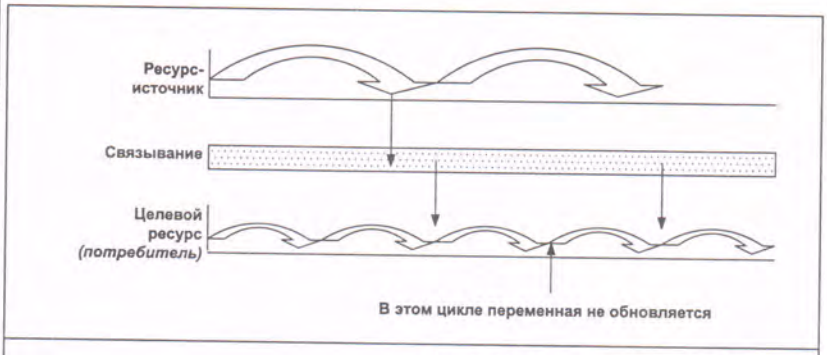

Рис. 2. Межресурсные связи переменных

Конфигурация относится к настройке исполнительной среды. Ресурсы необходимы для определения глобальных переменных, конфигурации и организации проекта, а также для наблюдения за изменением значений переменных. Задачи определяют схему планирования ассоциированных с ними программ в реальном времени. Это означает, что программы должны быть приданы задачам.

Декларирование задачи включает объявление ее имени, приоритета и условий, при которых задача начинает свое выполнение. К таким условиям можно отнести или интервал времени, по истечении которого задача вновь должна быть запущена; или передний фронт события, являющегося глобальной переменной. Каждой задаче может быть придано несколько программ, запускаемых задачей. Если задача выполняется в пределах текущего цикла, то и программы будут обработаны в границах этого цикла.

При наличии нескольких задач их выполнение подчиняется правилам:

– Выполняется та задача, для которой справедливы условия выполнения, то есть закончилось время цикла или выполнено условие.

– Если конкурируют несколько задач, то будет выполняться та, которая имеет больший приоритет.

– Если конкурируют несколько задач с одинаковым приоритетом, то будет выполняться та, которая требует большего времени.

Представленные понятия могут иметь иной смысл в программных системах типа SoftPLC. Так, в системе AlterSys фирмы CJ International (США) предлагаются следующие определения. Конфигурация – это программный объект, состоящий из одного или более ресурсов. Ресурс является набором программ и определений; он включает в себя параметры, группы переменных, программы, функции и функциональные блоки. В реальном времени ресурсу сопоставлен виртуальный глобальный объект Virgo (Virtual global object), который служит реализацией ресурса в исполнительной среде. Другими словами, Virgo представляет собой математическое обеспечение реального времени для одного ресурса одного проекта в исполнительной среде. Virgo исполняет код ресурса соответственно следующей схеме: опрашивает (сканирует) входные переменные; принимает значения связанных переменных; выполняет программные блоки; выдает значения связанных переменных; обновляет выходные регистры.

В случае, если были определены межресурсные связи переменных (рис. 2), принимаемые значения связанных переменных обновляются после опроса входных переменных; а выдаваемые другим ресурсам значения переменных посылаются перед обновлением выходных регистров. Связывание является направленной логической цепочкой между переменной ресурса-источника (производителя) и переменной целевого ресурса (потребителя). Связывание переменной V1 ресурса R1 с переменной V2 ресурса R2 означает, что V1 периодически копируется в V2, используя разделяемую память или сетевые механизмы обмена. Потребление связывающей информации со стороны другого ресурса осуществляется в начале цикла, а производство связывающей информации для другого ресурса происходит в конце цикла. Этот механизм напоминает устройство ввода-вывода.

Все входные переменные обновляются в начале каждого цикла. Это базовое поведение иногда изменяется с целью оптимизации в некоторых специфических драйверах ввода-вывода. Однако Virgo следит за тем, чтобы все входные переменные имели атрибут «read only». Цель состоит в стабильности образа входов. Переменная не изменяется в ресурсе-потребителе, пока ресурс-производитель не пошлет новое значение. Virgo не поддерживает «read only» доступ для потребляемых переменных. Однако рекомендуется объявлять тип потребляемых переменных как «read only» во избежание конфликтов между механизмом связывания и программными блоками.

Временная синхронизация двух Virgo посредством механизма связывания показана на рис. 3.

Структура проекта системы управления электроавтоматикой (клиентская часть). Проект содержит все необходимые компоненты программы управления электроав-

томатикой. Проект сохраняется в файле с тем же именем, что и проект. В проект входят следующие разделы: программные блоки POUs (Program Organization Units), типы данных, элементы визуализации, ресурсы и библиотеки.

POU является структурной единицей программы. В составе POU – функции, функциональные блоки и программы, которые могут быть дополнены действиями. Каждый POU состоит из двух частей: декларации и тела. Тело представляет собой программу контроллера, написанную на одном из языков стандарта IEC 61131-3: IL (Instruction List), ST (Structured Text), SFC (Sequential Function Chart), FBD (Functional Block Diagram), LD (Ladder Diagram). Для использования подобным образом построенных POUs в проект необходимо включить стандартную библиотеку. POUs могут вызывать другие POUs.

Функция представляет собой такой POU, которая будучи обработанной выдает в качестве результата всего лишь один элемент данных (возможно, из нескольких таких частей, как поля и структуры). В текстовых языках функция может быть оператором в выражении. При объявлении функции она должна получить тип. Это значит, что после имени функции следуют двоеточие и тип. Функции придается результат. Это значит, что имя функции используется как выходная переменная.

Функция не имеет внутренних условий. Это значит, что вызывая функцию с одними и теми же аргументами (входными параметрами) всегда получим один и тот же результат.

Функциональный блок представляет собой POU, который выдает во время работы один или более результатов. В отличие от функции, функциональный блок не возвращает значения. Можно создавать репродукции, то есть экземпляры функциональных блоков. Каждый экземпляр имеет собственный идентификатор (имя), а также структуру данных, располагающую входами, выходами и внутренними переменными. Экземпляры декларируются локально или глобально, между тем как имя функционального блока указывает на тип идентификатора. Функциональные блоки всегда вызываются через экземпляры. Только входные и выходные параметры экземпляра функционального блока доступны извне, но не внутренние переменные.

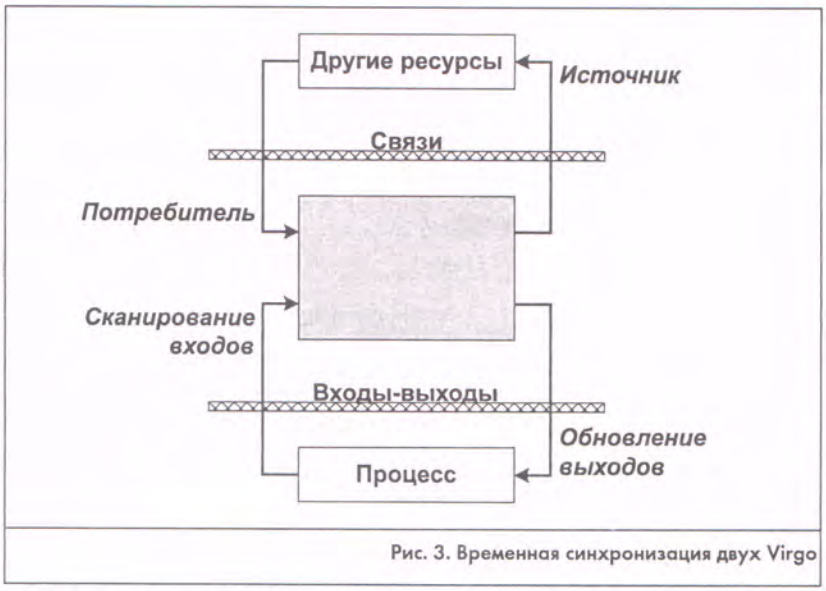

Рис. 3. Временная синхронизация двух Virgo

Декларационная часть функциональных блоков и программ может содержать декларации экземпляров. Декларации экземпляров для функций запрещены. Доступ к реализации функционального блока в POU, где он был продекларирован, ограничен, пока декларация не будет глобальной. Имя экземпляра функционального блока может быть использовано в качестве входа в функцию или функциональный блок.

Доступ к входным и выходным переменным функционального блока со стороны другого POU возможен путем создания экземпляра функционального блока и специфицирования желаемой переменной на основе следующего синтаксиса:

<Имя экземпляра>.<Имя переменной>

Программа представляет собой POU, который во время выполнения операций возвращает несколько значений. В пределах проекта программы глобально распознаваемы. Все значения переменных удерживаются с момента последнего цикла работы программы до начала следующего цикла. Программа может быть вызвана. Вызов программы в рамках функции запрещен. Не существует экземпляров программы. Если POU вызывает программу и в процессе ее работы значения переменных изменяются, они удерживаются к новому вызову программы, даже если в этом новом цикле программа вызвана из другого POU. В этом состоит разница с вызовом функционального блока. Там только значения в данном эк-

Рис. 4. Взаимоотношения ресурсов в системе Visual IOWorks

земпляре функционального блока изменяются. Эти изменения, следовательно, имеют значение, когда вызывается тот же экземпляр.

Действия могут быть определены по отношению к функциональным блокам и программам. Действия представляют собой развитие программирования и могут использовать другой язык. Каждому действию присваивается имя. Действие работает с данными из функциональных блоков и программ, которым оно принадлежит. Действие использует те же входные и выходные переменные и локальные переменные, какие использует и обычный экземпляр.

Разработчик может включить в свой проект серию библиотек, которые позволяют использовать POUs типы данных и глобальные переменные так, как если бы они были определены разработчиком.

Помимо стандартных типов пользователь может определить собственные. Могут быть созданы структуры, перечисляемые типы и ссылки.

Альтернативные структуры проекта в клиентской части. Система Visual IOWorks фирмы VMIC (США) предлагает объектно-ориентированный подход при разработке программ управления электроавтоматикой. Программа визуального программирования содержит следующие ресурсы: последовательности, диаграммы, страницы, библиотечные компоненты и символы. Рис. 4 показывает отношения этих ресурсов между собой.

Программа или ее модули состоят из отдельных последовательностей, которые при управлении становятся независимыми потоками. Независимость последовательностей-потоков означает, что в пределах модуля они выполняются самостоятельно и тем самым управляют исполнением модуля, которому принадлежат. Любая последовательность состоит из трех секций: секции инициализации, тела и закрывающей секции, каждая из которых представлена диаграммой.

Предположим, что последовательность определена или как циклическая, или управляемая по событию. Соответственно, в первом случае она будет работать непрерывно с заданной скоростью; во втором же случае последовательность будет запущена, если произойдет соответствующее событие. Множество периодических последовательностей одного и того же модуля могут быть запущены с разными скоростями.

Каждый модуль разрабатывают в визуальной форме с помощью Visual IOWorks, и все последовательности содержатся в этом файле. Это позволяет однократно объявить все определения и лучше организовать всю программу. В то же

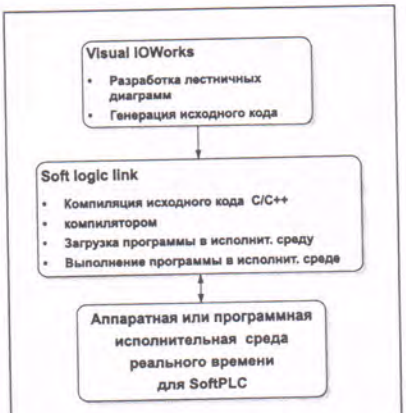

Рис. 5. Укрупненный алгоритм разработки и исполнения программы управления электроавтоматикой (фирма VMIC, США)

время каждая последовательность может быть отлажена независимо.

Visual IOWorks поддерживает графические языки стандарта IEC 61131-3. Диаграммы одной последовательности используют один из языков каждая; в пределах же файла можно использовать комбинации различных языков.

Три типа секций в последовательности означают три типа диаграмм: диаграмма инициализации, тело, закрывающая диаграмма. Диаграммы состоят из страниц. Инициализационная диаграмма выполняет функции конфигурации и инициализирует символы. Диаграммы тела описывают задачи последовательности, которые составляют ее цель. Закрывающая диаграмма деинициализирует входы-выходы, если это необходимо.

Рис. 6. Циклическая работа программы

Страница представляет собой плоскость, на которой изображается графическая программа. Страницы служат для логической структуризации программы, делая ее более читаемой и сопровождаемой.

Части являются графическими программными эквивалентами (в смысле IEC 61131-3) функций при C++ программировании. Каждая часть описывает специфическую задачу, выполняемую программой. Единообразные части сгруппированы в библиотеки.

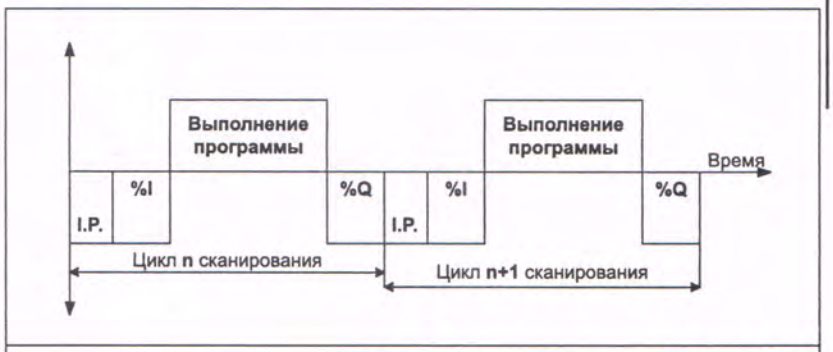

Рис. 7. Диаграмма циклической работы. Здесь: I.P. = системная работа; %I = чтение входов; %Q = обновление выходов

Символы – это переменные. После их объявления, они приписываются входам и выходам частей для передачи значений из одной в другую. Символы могут быть глобальными в том модуле, где они используются.

Укрупненный алгоритм разработки и исполнения программы управления электроавтоматикой фирмы VMIC показан на рис. 5.

Работа серверной части программы управления электроавтоматикой. Алгоритм выполнения программы в «нормальном» циклическом режиме показан на рис. 6. Содержание отдельных фаз цикла состоит в следующем.

В фазе системной работы программа осуществляет мониторинг контроллера (проверяет достаточность памяти, следит за сменой RUN/STOP, контролирует системные параметры и др.), обрабатывает запросы со стороны портов программирования и расширения. В фазе чтения входов обновляется внутренняя память в соответствии со статусом физических входов (%I). В фазе исполнения выполняется программа, написанная пользователем. В фазе обновления состояния физических выходов (%Q) обновляются из выходной памяти.

Цикл управления состоит в работе контроллера (процессор управляет системой, читает входы, выполняет программу и обновляет выходы) или остановке контроллера (процессор лишь управляет системой, читает входы и обновляет таблицу образов выходов; физические выходы не обновляются, пока системный бит %S=0).

Время цикла контролируется сторожевым таймером и не должно превышать определенного значения, например, 150 мсек. Иначе возникает ошибка, останавливающая контроллер. Существуют две ситуации:

Рис. 8. Периодическое управление

1. Время цикла сканирования меньше или равно настройке сторожевого таймера (150 мсек.). Это нормальная ситуация, при которой запускается очередной цикл сканирования.

2. Время сканирования больше настройки сторожевого таймера. Контроллер останавливается, загорается аварийная лампочка и устанавливается системный бит %S=1.

Рис. 9. Диаграмма периодической работы. Здесь: I.P. = системная работа; %I = чтение входов; %Q = обновление выходов

Диаграмма циклической работы показана на рис. 7. Другим вариантом выполнения программы служит периодическое управление (рис. 8). В этом случае чтение входов, выполнение программы, обновление выходов осуществляется периодически в соответствии с временем, установленным пользователем при конфигурации (например, от 2 до 150 мсек.). В начале цикла сканирования контроллера программный таймер устанавливает значение, заданное при конфигурации. Цикл сканирования должен завершиться до истечения этого времени. Если же это время превышено, системный бит %S устанавливается в единицу.

Время цикла контролируется сторожевым таймером и не должно превышать 150 мсек, иначе возникает ошибка, останавливающая контроллер. При этом могут возникнуть три ситуации:

1. Время сканирования меньше или равно периоду, установленному при конфигурации. Это нормальное управление. Следующий цикл сканирования начнется по истечении установленного периода.

2. Время, установленное при конфигурации, меньше времени сканирования, которое, в свою очередь, меньше времени сторожевого таймера. Системный бит %S устанавливается в 1, и пользователь ответственен за сброс его в нуль. Контроллер продолжает работать.

3. Время сканирования больше времени сторожевого таймера. Контроллер останавливается, загорается аварийная лампочка, а системный бит %S=1.

Диаграмма периодической работы показана на рис. 9.

Объектный подход при управлении электроавтоматикой. Рассмотрим этот подход в той форме, в какой он рекомендован фирмой AutomationX (США). Идея объектной ориентации состоит в попытке построить программные решения, приспособленные к многократному использованию. Для достижения этой цели программисту предлагаются три ключевых решения:

– Объединение функций и данных, которое позволяет разработчику создать специфическую функцию и метод присоединения данных к функции.

– Инкапсуляция, которая позволяет разработчику класса спрятать от конечного пользователя структуры данных, предъявляя только необходимые функции. Обычный пользователь может применять только те функции (иногда и данные), которые объявлены разработчиком класса как public.

– Наследуемая объектная функциональность. После создания объектов они существуют как экземпляры класса, наследуя структуру данных и функциональность класса. Пользователи включают объекты в собственные приложения, сопоставляя, таким образом, данным некий интеллект, как это определено в классах; и этот процесс называется встраиванием объектов, **Object Embedding**.

Существующие классы объединены в стандартные библиотеки, которые вполне исчерпывают потребности при автоматизации промышленных процессов. Кроме того, AutomationX Web Site содержит каталог классов, которые могут быть перегружены при необходимости. Другим эффективным методом является модификация существующих классов.

Объектный подход позволяет разработчику объединить в шаблоне класса самые разнообразные компоненты, необходимые при решении проблем автоматизации, в том числе: управление, визуализация, извещение об ошибках, накопление трендов, моделирование, доступ к базе данных, документирование.

Последовательность разработки реального приложения в контексте объектной ориентации состоит или в разработке классов, или в использовании готовых. Если класс создается заново, то это делается в следующей последовательности: определяют серверные и клиентские данные; разрабатывают функциональность класса; создают и параметризуют объекты; связывают объекты в контексте программы.

На первом шаге процесса проектирования необходимо создать представительный набор данных, которые будут повторяться в экземплярах классов (в объектах). Этот набор делится на два раздела: раздел сервера (для управления) и раздел клиента (для визуализации). Переменные в секции сервера принадлежат обычным типам (BOOL, INT, REAL, STRING и т. д.). Переменные в секции клиента могут принадлежать специальным типам, таким как FONT, COLOR, PIXELMA, и т. д.

Между клиентскими и серверными данными существует большая разница. Элементы серверных данных уникальны и постоянны в физической памяти. Элементы клиентских данных существуют во время визуализации объекта во время сессии редактора. Они дублируются в каждом экземпляре в период визуализации на экране. Функции клиентской группы имеют доступ к данным клиента и сервера. Функции серверной группы не должны иметь доступа к данным клиентской группы. Клиентские и серверные данные класса представлены на рис. 10.

Следующий шаг состоит в наполнении классов функциональностью; для этих целей используют библиотечные классы **Core Classes**, которые являются основными элементами AutomationX.

Все базовые классы, функциональные блоки и элементы структурированного текста языка ST могут быть включены в класс в рамках сессии редактора, и процессы системы приобретают соответствующие функции. Определенный набор библиотечных классов служит для сбора и оценки данных, для производственной логистики. Для функциональности за пределами стандарта IEC-классов можно добавлять C++ коды. Это необходимо в тех случаях, когда разрабатываемые алгоритмы сложны или разработчик класса хочет защитить собственное ноу-хау. Законченный класс состоит из группы файлов, которые организованы в виде библиотеки класса в директории AutomationX. Следующий шаг состоит в присвоении новому классу уникального имени. Прикладной инженер может изменить это имя в любое время.

Прикладной инженер должен разработать экземпляр класса. Это делается простым выбором класса в библиотеке AutomationX. При этом создается автономная копия выбранного класса, или объект.

Поведение объекта определяется параметрами, которые определяет разработчик класса. После параметризации процесс разработки завершается, верифицируется, а готовый файл сохраняется.

Теперь необходимо связать объекты с реальной программой. Не-

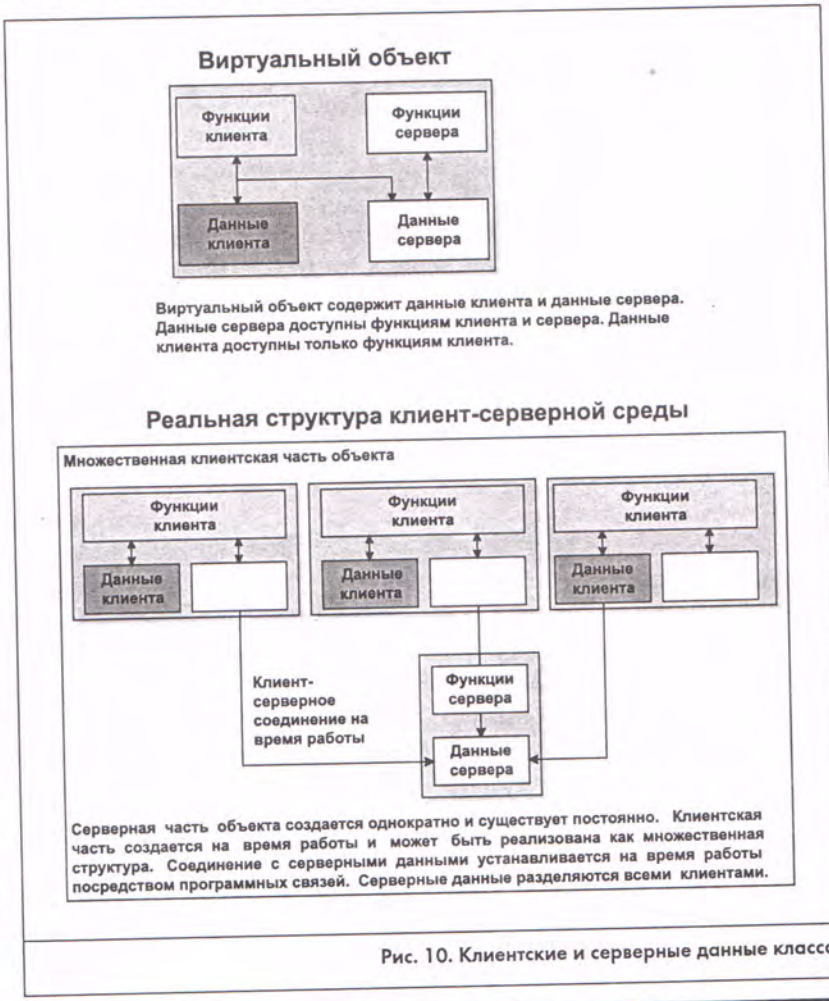

Рис. 10. Клиентские и серверные данные класса

Таблица 1

	Значение *(десятичное)*	Действие
0	1	Передача «с установлением связи». Если в данном кадре запрограммировано перемещение, то «установление связи» должно завершиться до начала движения.
1	2	Выход представляет собой битовый сигнал. Если в данном кадре запрограммировано перемещение, то выход должен состояться до начала движения.
2	4	Передача «с установлением связи». Если в данном кадре запрограммировано перемещение, то «установление связи» должно завершиться после конца движения.
3	8	Выход представляет собой битовый сигнал. Если в данном кадре запрограммировано перемещение, то выход должен состояться после конца движения.
4	16	Сброс сигнального бита перед началом движения. Если в данном кадре запрограммировано перемещение, то сброс должен состояться до начала движения.
5	32	Сброс сигнального бита после конца движения. Если в данном кадре запрограммировано перемещение, то сброс должен состояться после конца движения.
6	64	Зарезервировано.
7	128	Сигнал М-функции автоматически обнуляется в конце кадра, таким образом, он эффективен только в рамках одного кадра.

Таблица 2

Колонка	Значение
1	Номер М-функции
2	Битовая маска для правила поведения
3-12	Номер М-функции, которая должна быть сброшена

обходимо иметь в виду, что встраивание объектов позволяет комбинировать существующие объекты с библиотечными, с функциональными блоками, текстовыми элементами и т. д., выстраивая при этом некоторый сложный класс.

Особенности управления электроавтоматикой станков с ЧПУ. Первая особенность состоит в том, что в стандарте DIN66025 имена циклов электроавтоматики устанавливают вспомогательные М-функции. Другая особенность заключается в наличии интерфейса NC <-> PLC между математическим обеспечением ЧПУ и контроллером (программным модулем управления электроавтоматикой). Третья особенность относится к SoftPLC, когда программное обеспечение ЧПУ и контроллера располагается в единой исполнительной среде [1-3].

М-функции дискретны, они включаются и выключаются, активизируются и деактивизируются. При этом, передача каких-либо параметров не предусмотрена; но эта проблема решается иным об-

Таблица 3

Группа	М-функции	Семантика
Группа завершения программы	M00	Безусловный останов программы. Программа дорабатывается до конца кадра. Шпиндель останавливается, подача охлаждающей жидкости прекращается. После повторного нажатия кнопки ПУСК отработка управляющей программы возобновляется. Но перед этим включается шпиндель и подача охлаждающей жидкости. Предусмотрена предустановка времени разгона шпинделя.
	M01	Условный останов, работает как и M00, но при условии подтверждения клавишей на панели оператора.
	M02/M30	Эти инструкции завершают отработку управляющей программы. Шпиндель останавливается, подача охлаждающей жидкости прекращается.
	M99	Конец программы, как и M30, но с дополнительным отводом исполнительных органов в безопасную позицию. Безопасная позиция определяется машинными параметрами
Группа управления шпинделем	M03/M04	Включение вращения шпинделя по/против часовой стрелки. После предустановленного времени разгона начинается отработка других функций кадра. Функция деактивируется либо командой M05 (останов шпинделя), либо командой завершения программы.
	M05	Останов шпинделя, который произойдет, когда все инструкции кадра выполнены.
	M19	С помощью функций M19 S<> можно остановить шпиндель в ориентированном положении, причем S-функция указывает угол поворота в градусах. Например: M19 S90. Конечно, это возможно, если используется следящий привод управления шпинделем.
Группа функций охлаждения	M07	Включение функции охлаждения с идентификатором 1. Функция работает перед началом выполнения кадра и выключается либо функцией M09 (выключение охлаждения), либо функцией завершения программы.
	M08	Включение функции охлаждения с идентификатором 2.
	M09	Выключение охлаждения после того, как все функции кадра отработаны.
Группа смены инструмента	M06	Замена инструмента. Процедура состоит в следующем: – Шпиндель выключается; – Исполнительные органы уходят в безопасную позицию; – Запускается машинно-зависимая процедура замены инструмента; – Если замена инструмента осуществляется вручную, то завершение этого процесса должно быть подтверждено. После окончания замены исполнительные органы остаются в безопасном положении, а шпиндель продолжает быть выключенным.

разом (с помощью S, T, H функций). Возможны два способа передачи сигнала контроллеру: «быстрыми битовыми сигналами» или «с установлением связи».

Если управляющие М-функции не требуют обратной связи и ожидания выполнения, то можно воспользоваться быстрыми сигнальными битами. Это предполагает присутствие группы битов в интерфейсе NC <-> PLC. Каждая М-функция представлена одним сигнальным битом, который устанавливается и сбрасывается системой ЧПУ. В этом случае система ЧПУ не ждет и не рассматривает никаких сигналов со стороны контроллера. Сигнальные биты остаются в канале интерфейса, пока они не сброшены или система ЧПУ не перезапущена. Сброс может произойти автоматически в конце кадра или при вызове другой М-функции.

Функции, требующие обратной связи, должны обрабатываться с использованием дуплексного обмена сигналами между NC <-> PLC. Для этих целей используют вспомогательный интерфейс NC <-> PLC канала. Каждая М-функция регистрируется здесь независимо со своим номером и запрашиваемым сигнальным битом. Выполнение очередной функции возможно лишь после того, как контроллер подтвердит завершение выполнения М-функции, и сигнальный бит вернется в нейтральное состояние (нет запроса, нет подтверждения). Операция, выполняемая под управлением ЧПУ, координируется, таким образом, с работой оборудования под управлением контроллера. Поскольку этот тип М-функций работает синхронно, только одна М-функция «с установлением связи» может быть в данный момент активна в управляющей программе.

Последующие «правила поведения» предписывают установку битов в управляющей маске для каждой М-функции (таблица 1).

Все М-функции описаны в конфигурационном файле. Каждая М-функция описана в этом файле в своей строке, как показано в таблице 2.

Можно комбинировать правилами поведения, чтобы добиться нужной функциональности. Не все комбинации имеют смысл, а некоторые могут привести к ситуации, когда NC будет ожидать события, которое никогда не произойдет. Поэтому к правилам следует относиться очень внимательно.

Кодирование М-функций частично определено в стандарте DIN66025, а частично устанавливается производителем оборудования. Помимо команд управления шпинделем (M03, M04, M05, M19) и команд завершения программы (M02, M30, M99), все другие М-функции могут быть свободно использованы станкостроителями. В одном кадре используют до восьми М-функций. Те из них, которые не зафиксированы в стандарте DIN66025, должны быть первыми.

Семантика М-функций представлена в таблице 3.

Заключение. Системы управления электроавтоматикой построены соответственно клиент-серверной модели. Размещение клиентской и серверной частей в одном персональном компьютере послужило основой концепции SoftPLC. Многочисленные современные системы программирования придерживаются стандарта IEC 61131-3; при этом ориентированы на общепромышленную электроавтоматику и имеют мощную инструментальную поддержку, в которой доминирует объектный подход. В серверной части отдельные задачи реального времени работают циклически, периодически или по событию. Существуют предложения, согласно которым задачи разнесены по своим потокам. Особенности управления электроавтоматикой станков по типу SoftPLC заключаются в том, что задачи SoftPLC квазипараллельны задачам ЧПУ и работают в одной и той же исполнительной среде. Другая особенность состоит в том, что циклы управления электроавтоматикой вызываются из управляющей программы. Программирование этих циклов, по-видимому, требует своей объектно-ориентированной поддержки.

ЛИТЕРАТУРА:

1. Yoshinory Tsujido. The trend towards open-system controllers // Mitsubishi Electric Advance. 1996, September. – P. 23-24.

2. J. Stenerson. Fundamentals of Programmable Logic Controllers, Sensors and Communications. – Prentice Hall, ISBN 0137461240. 1998. – P. 562

3. G. Dunning. Introduction to Programmable logic Controllers. – Delmar Publishers, – ISBN 0827378661. 1998. – P. 433.