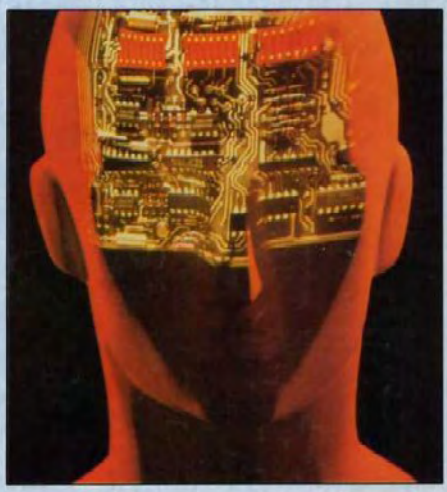
Новая Университетская Библиотека

В.Л. Сосонкин
Г.М. Мартинов

СИСТЕМЫ ЧИСЛОВОГО ПРОГРАММНОГО УПРАВЛЕНИЯ



УЧЕБНОЕ ПОСОБИЕ

Новая
Университетская
Библиотека

Посвящается 75-летию
МГТУ «СТАНКИН»

В.Л. Сосонкин, Г.М. Мартинов

СИСТЕМЫ ЧИСЛОВОГО ПРОГРАММНОГО УПРАВЛЕНИЯ

Учебное пособие для студентов высших учебных заведений, получающих образование по направлению 550200 «Автоматизация и управление», специальности 210200 «Автоматизация технологических процессов и производств» и магистерской программе 550207 «Распределенные компьютерные информационно-управляющие системы»

Москва
Логос
2005

УДК 004.4
ББК 32.965.7
С54

Рецензенты

Ю.А. Купеев, кандидат технических наук, профессор
Е.Н. Ивашов, доктор технических наук, профессор

Сосонкин В.Л., Мартинов Г.М.

С54 Системы числового программного управления: Учеб. пособие. – М.: Логос, 2005. – 296 с.

ISBN 5-98704-012-4

Представлены архитектурные решения локальных систем числового программного управления, дан анализ вариантов их интеграции на системном и прикладном уровнях. Рассмотрены особенности новейшего поколения систем ЧПУ. Основное внимание уделено модульному построению однокомпьютерной системы ЧПУ и принципам взаимодействия модулей в рамках открытой архитектуры. Представлены все разделы прикладного программного обеспечения систем ЧПУ. Показано, каким образом понятия предметной области (интепретатор, интерполятор, языковый процессор и др.) соотносятся с понятиями программирования (объектный тип, атрибут, операция и др.). Охарактеризованы функциональные возможности современных систем ЧПУ.

Для студентов высших учебных заведений, обучающихся по специальности 210200 «Автоматизация технологических процессов и производств», направлению 550200 «Автоматизация и управление» и магистерской программе 550207 «Распределенные компьютерные информационно-управляющие системы». Может быть полезна преподавателям, аспирантам и специалистам.

УДК 004.4
ББК 32.965.7

ISBN 5-98704-012-4

Оглавление

Список сокращений

ACS	–	Axes Coordinate System
ADS	–	Automation Device Specification
AP	–	Application Protocol
API	–	Application Programming Interface
ATL	–	Active Template Library
CAD	–	Computer-Aided Design
CAM	–	Computer-Aided Manufacturing
CAN	–	Controller Area Network
CAPP	–	Computer-Aided Process Planning
CiA	–	CAN in Automation
COM	–	Component Object Model
DA	–	Data Access
DB	–	Data Base
DLL	–	Dynamic Link Library
DNA	–	Distributed IntraNet Application
DTD	–	Document Type Declaration
ERP	–	Enterprise Resource Planning
FBD	–	Functional Block Diagram
GUID	–	Globally Unique Identifier
HAL	–	Hardware Abstraction Layer
IDL	–	Interface Definition Language
IGES	–	Initial Graphics Exchange
IL	–	Instruction List
IPD	–	InterPolator Data
LD	–	Ladder Diagram
MAP	–	Manufacturing Automation Protocol
MCS	–	Machine Coordinate System
MIO	–	Main Input-Output
MES	–	Manufacturing Execution System
MFC	–	Microsoft Foundation Classes
MMI	–	Man-Machine Interface
MMS	–	Manufacturing Message Specification
MRP	–	Manufacturing resource Planning
NC	–	Numerical Control
NCS	–	Numerical Control System
OLE	–	Object Linking and Embedding

OOC – Object Oriented Channel
OPC – OLE for Process Control
OPI – OEM Program Interface
PC – Personal Computer
PCNC – Personal Computer Numerical Control
PCS – Program Coordinate System
PLC – Programmable Logic Controller
PMAC – Programmable Multi-Axes Controller
POUs – Program Organization Units
PWM – Pulse Width Modulation
RFC – Remote Function Call
RTAPI – Real Time Application Interface
PTSS – Real-Time Sub-System
RTTI – Run-Time Type Information
RTX – Real Time Extension
RTX – Real Time eXtension
SCADA – Supervisory Control and Data Acquisition
SFC – Sequential Function Chart
SM – Special Marker
SP – Service Pack
ST – Structured Text
STEP – Standard for the Exchange of Product Model Data
TCS – Tool Coordinate System
UML – Unified Modeling Language
VMD – Virtual Manufacturing Device
WCS – Workpiece Coordinate System
WSN – Workplane for Setting Null
XML – Extensible Markup Language
XSDL – XML Schema Definition Language
XSL – Extensible Stylesheet Language

Предисловие

Опыт преподавания ряда профилирующих дисциплин на кафедре компьютерных систем управления показал, что полностью отсутствует современная литература, поддерживающая проблематику числового программного управления. К этим дисциплинам относятся: «Программное обеспечение систем управления», «Автоматическое управление процессами и системами», «Структура и математическое обеспечение систем управления», «Распределенные системы управления». Дисциплины принадлежат типовым учебным планам по специальности 210200 «Автоматизация технологических процессов и производств»; по направлению 550200 «Автоматизация и управление»; по магистерской программе 550207 «Распределенные компьютерные информационно-управляющие системы».

У этих дисциплин общим является предметная область и понятийный аппарат, т.е. некоторое единое ядро, которое рассматривается в разных дисциплинах с различных позиций и сторон. Так родилась идея этой книги, которая состоит в том, чтобы содержательно и методически поддержать все четыре дисциплины, используя новейшие подходы и знания, приобретенные авторами, главным образом, в процессе их собственной научной работы.

Этой книге предшествовало написание более двух десятков статей в наиболее популярных технических журналах, пока, наконец, все необходимые темы оказались охваченными. Затем был создан двуязычный сайт www.ncsystems.ru, на котором (в том числе) были представлены эти статьи, но не в систематизированной форме. Тем не менее статьи вызвали большой интерес, о чем свидетельствует непрерывный рост рейтинга сайта. Однако создание задуманного учебного пособия оставалось по-прежнему актуальным, учитывая большой контингент студентов по специальности 210200, направлению 550200 и магистерской программе 55027.

Введение

В течение последних десяти лет у нас в стране не было сколько-нибудь серьезных и полных публикаций, посвященных проблематике числового программного управления (ЧПУ). Между тем за это время в области ЧПУ произошли кардинальные изменения, затрагивающие спектр функциональных возможностей, аппаратную платформу и системные средства, архитектуру и состав прикладного математического обеспечения. Доминирующие позиции заняла концепция открытых систем ЧПУ, построенных на базе персонального компьютера (PCNC – Personal Computer Numerical Control). Другими словами, современные системы ЧПУ – это совсем не то представление, которое может возникнуть в воображении читателей монографий и учебников, написанных в 80-х и начале 90-х годов прошлого века. Более того, мы уверены, что подобное представление будет превратным.

В этой связи был предпринят проект, нашедший свое отражение на нашем сайте www.ncsystems.ru и в предлагаемой вниманию читателя книге. Проект затрагивает все основные проблемы современного ЧПУ. Реализация проекта была бы невозможной без теоретических исследований и практического опыта авторов, который сложился на основе их собственных разработок и сотрудничества с ведущими фирмами Запада за последние 12 лет.

Книга содержит пять глав.

Первая глава посвящена обзору архитектурных решений локальных систем ЧПУ, а также анализу вариантов их интеграции на системном уровне (OLE for Process Control – OPC) и прикладном уровне (Standard for Exchange of Product model data – STEP). Читателю предлагается ознакомиться с «архитектурным спектром» систем ЧПУ и функциональными возможностями, вытекающими из выбора конкретного варианта архитектуры. Наше предпочтение отдается компьютерному варианту. Новейшие тенденции состоят в использовании концепций «открытого управления» и включении систем ЧПУ в структуру «жизненного цикла» производства изделий (STEP).

Вторая глава посвящена изложению ключевых особенностей новейшего поколения систем ЧПУ. К их числу отнесены модульная организация систем ЧПУ типа PCNC, проблемы реального времени и внутренней коммуникации, архитектура управления электроавтоматикой, проблема создания удаленных терминалов систем ЧПУ и проблема перехода к новому

поколению систем ЧПУ, соответствующих стандарту STEP. Основное внимание уделено модульному построению однокомпьютерной системы ЧПУ и принципам взаимодействия модулей в рамках открытой архитектуры.

Особо выделены те модули, которые работают в реальном времени и требуют соответствующей системной поддержки. В их числе рассмотрен и новый программный модуль управления электроавтоматикой без привлечения аппаратуры программируемых контроллеров. Коммуникационная среда, являясь глобальным сервером системы ЧПУ, принимает на себя проблему интеграции всех модулей и проблему межмодульного взаимодействия, в том числе и в рамках распределенного управления.

К другим актуальным проблемам отнесены разработка модели удаленных терминалов на основе Java-апплетов и построение такой архитектуры ЧПУ, которая поддерживает стандарт STEP-NC (ISO 14649). Этот стандарт приведет в ближайшее время к появлению систем ЧПУ очередного поколения.

В *третьей главе* представлены все разделы прикладного программного обеспечения систем ЧПУ, т.е. так называемые задачи управления: геометрическая, логическая, терминальная, диагностическая. Каждая отдельная задача рассмотрена на том уровне, который отвечает концепции построения «продвинутой» системы ЧПУ. Геометрическая задача является центральной и отвечает за формообразование. Логическая задача занимается управлением электроавтоматикой; здесь показано наиболее современное ее решение по типу виртуального контроллера, т.е. без привлечения специальных аппаратных средств.

Терминальная задача организует диалоговый интерфейс с оператором. В рамках построения диагностической задачи в качестве составной части прикладного программного обеспечения рассмотрены логический анализатор и осциллограф.

В *четвертой главе* показано, что произошло радикальное изменение взглядов на процедуру разработки прикладного математического обеспечения ЧПУ. Это связано с использованием объектно-ориентированного программирования, компонентного подхода и средств инструментальной поддержки проектных работ. Объектные и компонентные модели математического обеспечения систем управления имеют свою специфику. В этой связи показано, каким образом понятия предметной области (интерпретатор, интерполятор, языковый процессор, групповой интерпретатор, канал) соотносятся с понятиями программирования (объектный тип, атрибут, операция, наследование, полиморфизм, интерфейс и компонент).

Рассмотрены объектно-ориентированные модели систем ЧПУ типа PCNC с открытой архитектурой. Для поддержания разработки систем с

открытой архитектурой предложены стандартные и оригинальные инструментальные средства. Компонентный подход проиллюстрирован на примере контроллера привода. Очерчена рекомендуемая область компонентного проектирования и рекомендованы полезные инструментальные средства.

Пятая глава посвящена обзору функциональных возможностей современных систем ЧПУ, которые прямым или косвенным образом отображены в различных документах пользователя. Так, приведена структура руководства по программированию с описанием фазового пространства технологической машины и общим представлением относительно версий кода ISO-7bit. Раскрыто понятие конфигурации системы ЧПУ, суть которого состоит в настройке системы на конкретный объект управления с помощью глобальных переменных, называемых машинными параметрами. Изложена современная методика программирования систем ЧПУ с наиболее полным набором G-функций. Наконец, описана методика разработки управляющей программы ЧПУ соответственно стандарту ISO 14649 STEP-NC.

Глава 1.
Классификация систем управления

Если сравнивать системы ЧПУ лишь по их внешним «паспортным» характеристикам, то трудно объяснить их функциональное разнообразие и их несовместимость при попытках интеграции различного уровня в пределах одного и того же предприятия. Для понимания причин несовместимости необходимо обратиться к внутренней организации и структуре систем ЧПУ, и здесь полезной оказывается классификация архитектурных решений. Эта классификация позволяет проследить эволюцию ЧПУ, которая привела к построению систем управления на базе персонального компьютера. Выбор архитектурного решения определяет возможность (или невозможность) интеграции систем ЧПУ. Говоря об интеграции систем управления на программно-аппаратном уровне, следует обратить внимание на один из перспективных вариантов, использующих стандарт OPC. С другой стороны, единственным вариантом интеграции в рамках технологической среды предприятия и полного жизненного цикла производства является внедрение стандарта STEP.

1.1. Современный мировой уровень архитектурных решений в области ЧПУ

Представлен и проиллюстрирован анализ архитектуры систем ЧПУ. Приведена классификация, указывающая на сосуществование на рынке ЧПУ пяти архитектурных вариантов. Показано, что в спектре архитектурных решений наиболее уверенные позиции занимает концепция PCNC. Самая значительная тенденция состоит в развитии и реализации идей открытой архитектуры ЧПУ, которая предоставляет конечному пользователю широкие возможности для внедрения в систему ЧПУ собственных функций.

Классифицированные архитектурные варианты сведены в табл. 1.

Классические системы CNC (первый вариант) до сих пор выпускаются лишь фирмами с богатой традицией производства высококачественной собственной микроэлектронной аппаратуры. Но и эти фирмы под давлением

Таблица 1. Классификация архитектурных решений систем ЧПУ

<table>
<tr><th></th><th>CNC</th><th>PCNC-1</th><th>PCNC-2</th><th>PCNC-3</th><th>PCNC-4</th></tr>
<tr><td>Персональный компьютер</td><td rowspan="2"></td><td colspan="3">Интерфейс оператора</td><td>Интерфейс оператора. Ядро ЧПУ. Программно-реализованный контроллер электроавтоматики.</td></tr>
<tr><td>Встроенный одноплатный компьютер</td><td colspan="2" rowspan="2"></td><td>Ядро ЧПУ. Программно-реализованный контроллер электроавтоматики.</td><td></td></tr>
<tr><td>Интерфейс</td><td colspan="2">Коммуникационный интерфейс</td><td rowspan="2"></td></tr>
<tr><td>Второй компьютер</td><td colspan="2"></td><td>Ядро ЧПУ. Программно-реализованный контроллер электроавтоматики</td></tr>
<tr><td>Специальный процессорный модуль</td><td>Интерфейс оператора. Ядро ЧПУ. Внешний контроллер электроавтоматики.</td><td>Ядро ЧПУ. Одноплатный контроллер электроавтоматики</td><td colspan="3"></td></tr>
<tr><td>Интерфейс</td><td colspan="2">Управление приводами и электроавтоматикой</td><td colspan="3">Периферийные шины следящих приводов и электроавтоматики</td></tr>
<tr><td>Объекты управления</td><td colspan="2">Локальные объекты</td><td colspan="3">Сетевые объекты</td></tr>
</table>

конечных пользователей, желающих иметь гибкий интерфейс оператора, предлагают модификацию с персональным компьютером в качестве терминала (второй вариант). По многим причинам [1] первые системы типа PCNC относились к двухкомпьютерной архитектуре (третий вариант); они и сегодня очень популярны и наиболее широко распространены. Несколько позднее появились системы PCNC, ядро которых реализовано на отдельной плате, устанавливаемой в корпусе промышленного персонального компьютера (четвертый вариант). Наконец, по мере повышения мощности микропроцессоров все большее распространение получает однокомпью-

терный вариант системы PCNC (пятый). Все варианты отражают суммарный опыт разработчиков систем ЧПУ и перспективные тенденции. В этой связи их рассмотрение достаточно поучительно, в особенности для тех, кто занимается разработкой новых моделей у нас в стране.

1.1.1. Системы CNC и PCNC-1

Семейство систем фирмы NUM (Франция, в составе концерна Schneider, Германия) построено по принципу многопроцессорных CNC-систем, т.е. с ЧПУ-процессором, процессором программируемого контроллера автоматики и графическим процессором (рис. 1). Система NUM может быть оснащена пассивным терминалом или промышленным компьютером с операционной системой Windows 98. Семейство представлено компактными, а так-

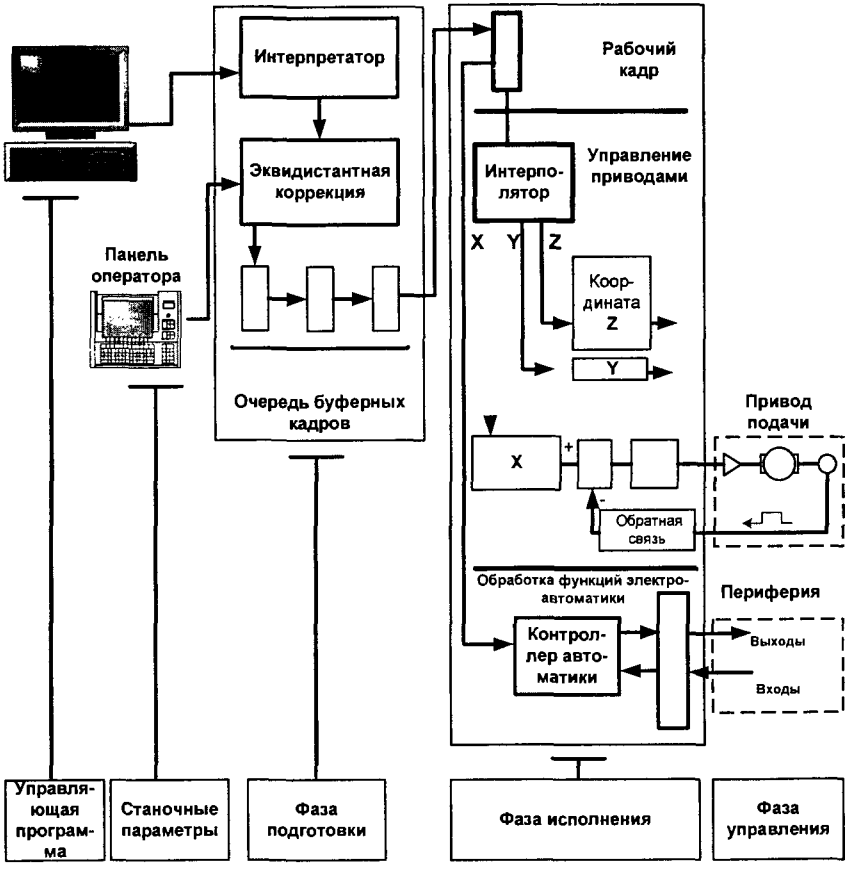


Рис. 1. Архитектура системы ЧПУ класса CNC фирмы NUM

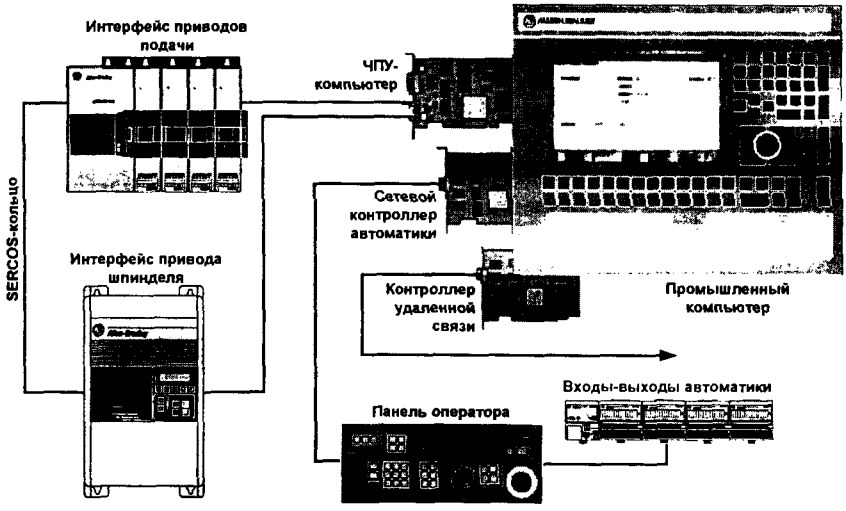


Рис. 2. Архитектура системы ЧПУ класса PCNC-1 фирмы Allen-Bradley

же и модульными версиями, которые различаются числом координат (и возможностью формировать независимые каналы ЧПУ из координатных групп), использованием традиционных аналоговых или автономных цифровых следящих приводов, подключенных к оптоволоконной сети, а также числом входов-выходов электроавтоматики и использованием удаленных (сетевых) входов-выходов.

Вычислительная мощность систем NUM исключительно высока, и этим объясняется широкий набор их функциональных возможностей. Так, предусмотрены сплайновый и полиномиальный (до пятого порядка) алгоритмы интерполяции, пяти–девятикоординатная интерполяция, пятикоординатная коррекция инструмента, одновременная работа по двум различным управляющим программам, 3D-графика и др. В системах с терминальным компьютером возможна адаптация интерфейса оператора к запросам конечных пользователей, диалоговое программирование с помощью инструментальных систем PROGRAM_MILL и PROGRAM_TURN.

Фирма Allen Bradley в составе концерна Rockwell (США) выпускает широкое семейство систем ЧПУ: от традиционной CNC (модель 9/440) до систем CNC с персональным компьютером в качестве терминала (модель 9/260(290)) и систем класса PCNC (модель 9/PC). Последняя модель (рис. 2) выполнена по вполне классической схеме: специализированный промышленный компьютер с Windows NT операционной системой и возможностью разрабатывать пользовательские приложения на Visual Basic (функции прикладного интерфейса API опубликованы); PCI – одноплатный ЧПУ-3

компьютер, выполняющий все функции ядра, включая программно-реализованный контроллер электроавтоматики.

Программирование и редактирование контроллера осуществляются через общий для всей системы терминал. Программируемый контроллер имеет собственную сеть (и сетевую плату).

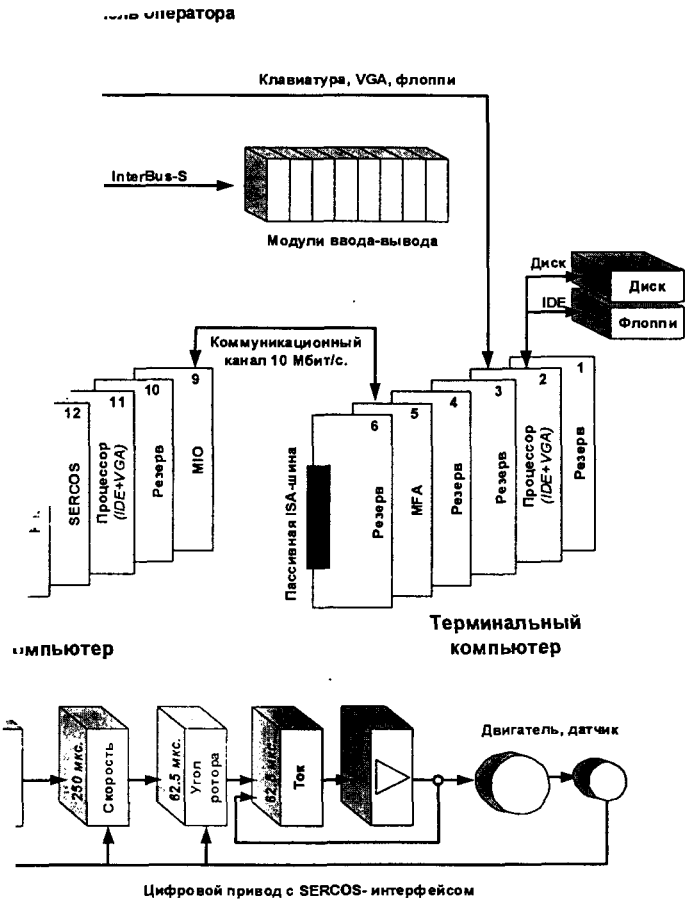
1.1.2. Системы PCNC-2

К этому классу принадлежат системы фирм ANDRON и BoschRexroth (Германия). Система ЧПУ фирмы ANDRON относится к полному двухкомпьютерному варианту. Ее структура представлена на рис. 3 в виде набора модулей: терминального компьютера, ЧПУ-компьютера, панели оператора и монитора, удаленных входов-выходов программируемого контроллера, одной или нескольких групп цифровых (SERCOS) приводов подачи и главного привода. Аппаратура системы практически полностью состоит из покупных компонентов и плат. В силу этого обстоятельства фирма ANDRON не скрывает деталей аппаратной реализации, и эта реализация весьма представительна для двухкомпьютерных версий систем ЧПУ других фирм.

В состав терминального компьютера входят: материнская плата с Celeron-процессором и интегрированными контроллерами SCSI, VGA, TFT, IDE; многофункциональная интерфейсная плата MFA с памятью CMOS-ROM (связь с внешним модемом; транспьютерный контроллер коммуникационного канала, связывающего терминальный и ЧПУ-компьютеры). Все платы установлены на пассивной ISA-шине, при этом предусмотрена установка дополнительных (по заказу) резервных плат: внутреннего модема, сетевой платы, SCSI-платы. Для специальных задач возможна установка PCI-плат.

В состав ЧПУ-компьютера входят: материнская плата с Celeron-процессором; плата MIO (Main Input-Output) поддержки как коммуникационного интерфейса с терминальным компьютером (со скоростью 10 Мбит/с), так и интерфейса маховичка ручного перемещения; плата программируемого контроллера с интерфейсом InterBus-S (с циклом 4 мс для 1024 входов-выходов); одна или несколько плат SERCOS-интерфейса (с микросхемой SERCON410-B). Все платы установлены на пассивной ISA-шине. Каждый SERCOS-интерфейс обслуживает (с периодичностью 0,5 мс) одну группу из трех автономных приводов подачи и одного привода шпинделя. Приводы одной группы включены в кольцевую оптоволоконную сеть.

В платформе системы ЧПУ фирмы ANDRON аппаратный уровень расположен под операционной системой Windows NT в терминальном компьютере и оригинальной операционной системой реального времени в ЧПУ-компьютере. На прикладном уровне терминальный компьютер открыт для разнообразных приложений и специальных диалогов конечного пользова-



...ектура системы ЧПУ класса PCNC-2 фирмы ANDRON

орые можно назвать CAM-приложениями. Для построения CAM-ений предусмотрен инструментальный язык ANLOG-C, обеспечи-ли доступ к функциям ядра в ЧПУ-компьютере.

-ма ЧПУ (Typ3.osa) фирмы BoschRexroth построена на основе зысокопроизводительных компьютеров (классический двухкомпь-ый вариант) и обладает исключительно мощным набором функ-ии. 4). Терминальный компьютер имеет операционную систему ows NT, а ЧПУ-компьютер – операционную систему UNIX. Связь рационных сред осуществляется с помощью протоколов TCP/IP, что опускает удаленное размещение терминала и работу нескольких терми-

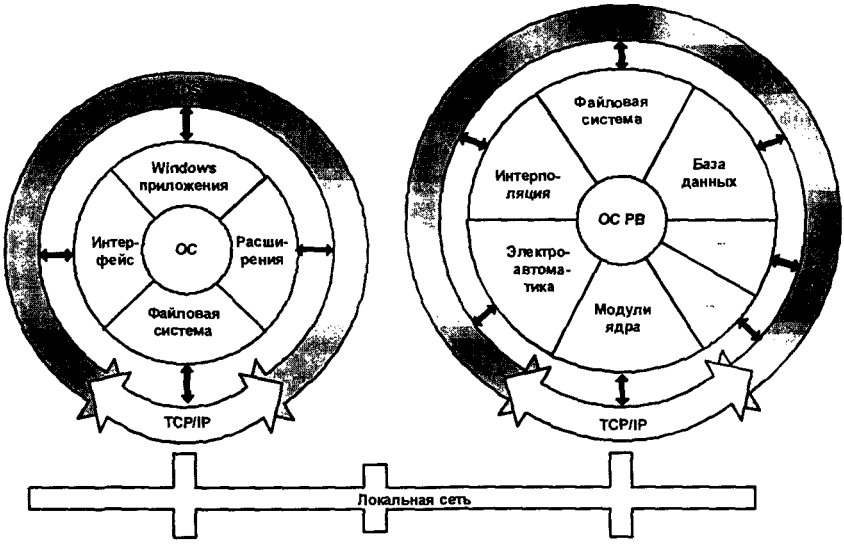


Рис. 4. Архитектура системы ЧПУ класса PCNC-2 фирмы BoschRexroth: ОС – операционная система; ОС РВ – операционная система реального времени

налов с одним ЧПУ-компьютером. В свою очередь ЧПУ-компьютер предполагает многоканальную работу более чем с одной управляющей программой. Прикладное математическое обеспечение терминального компьютера и прикладное математическое обеспечение ядра в ЧПУ-компьютере окружены оболочкой из нескольких сот интерфейсных API-функций (Application Programming Interface), которые предоставляют конечным пользователям возможность разрабатывать собственные приложения и расширения. В оболочку терминального компьютера включена мощная DLL-библиотека NCS (Numerical Control System) классов объектов, «покрывающая» API-функции, делающая разработку дополнительных приложений более простой и комфортной. В остальном состав прикладного математического обеспечения традиционен; впрочем, можно отметить хорошо проработанный программно-реализованный контроллер электроавтоматики и несколько очень интересных приложений. Среди них – отладчик высокоуровневых управляющих программ, логический анализатор для удаленного контроля программируемых контролеров, осциллограф для анализа динамики следящего привода, в том числе и с помощью рассчитываемых здесь же частотных характеристик.

1.1.3. Система PCNC-3

Типичным представителем систем этого класса является система фирмы DeltaTau (Великобритания). Она относится к двухкомпьютерному ва-

рианту, но такому, при котором ЧПУ-компьютер выполнен в виде отдельной платы PMAC (Programmable Multi-Axes Controller), устанавливаемой на ISA (или PCI)-шине терминального персонального компьютера (рис. 5). Терминальный компьютер с Windows NT операционной системой выполняет классические функции терминальной задачи и функции интерпретатора управляющих программ. Одноплатный ЧПУ-компьютер PMAC (процессор Motorola 56300) решает геометрическую и логическую задачи [2, 3], выполняя функции интерполятора, контроллера управления приводами (подачи и шпинделя), программно-реализованного контроллера электроавтоматики.

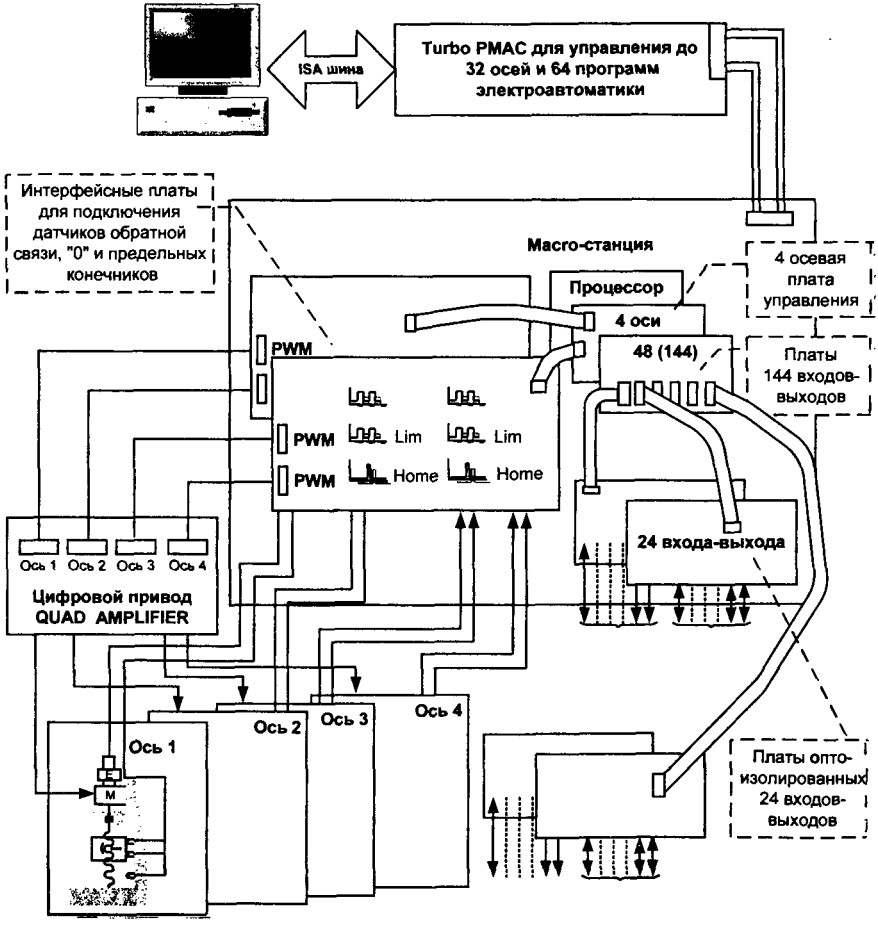


Рис. 5. Архитектура системы ЧПУ класса PCNC-3 фирмы DeltaTau:
PWM – Pulse Width Modulation, широтно-импульсная модуляция;
Lim (Limit) – ограничители; Home – нулевая точка

Интерполятор обеспечивает все виды интерполяции (включая сплайновую), разгоны и торможения, опережающий просмотр кадров Look Ahead, циклическое формирование управляющих воздействий с периодом 440 мкс (в этом же периоде в фоновом режиме работает и контроллер электроавтоматики).

Контроллер приводов способен управлять 32 координатными осями, сгруппированными в 16 координатных систем; он принимает сигналы позиционных датчиков обратной связи, замыкает позиционные контуры, выполняет функции ПИД-регулятора, имитирует в цифровом виде сигналы обратной связи по скорости, вырабатывает (в цифровом виде) широтно-импульсный сигнал для приводов подачи и сигнал ±10В для привода главного движения. Программно-реализованный контроллер электроавтоматики поддерживает параллельное управление 64 циклами электроавтоматики.

Выходные сигналы (для управления приводами и электроавтоматикой) поступают в кольцевой оптоволоконный канал (со скоростью передачи данных 125 Мбит/с) для дистанционного управления своими объектами. Принимающим устройством служит интеллектуальный периферийный терминал Macro-станция (Motion and Control Ring Optical). Допустимо включение в кольцо нескольких таких терминалов. Терминал замыкает скоростные контуры восьми приводов и принимает сигналы ограничителей рабочей зоны и датчиков нулевых точек координатных систем (в блоках ACS), формирует сигналы управления двигателями любого типа (асинхронными, постоянного тока и др.) с помощью блока Quad Amplifier (для управления четырьмя двигателями общей мощностью до 25 кВт). Другая функция периферийного терминала – управление электроавтоматикой через модули оптоизолированных входов-выходов.

Набор модулей фирмы DeltaTau (PMAC и Macro) ориентирован на построение собственных систем ЧПУ у конечных пользователей, на долю которых остается разработка терминальной задачи, и интерпретатора в среде промышленного персонального компьютера. Однако сами модули являются для конечного пользователя «черными ящиками» и их архитектура закрыта.

1.1.4. Системы PCNC-4

Система ЧПУ фирмы Beckhoff (Германия) демонстрирует яркий пример чисто однокомьютерной архитектуры PCNC, в рамках которой все задачи управления (геометрическая, логическая, терминальная) решены чисто программным путем, без какой-либо дополнительной аппаратной поддержки (рис. 6).

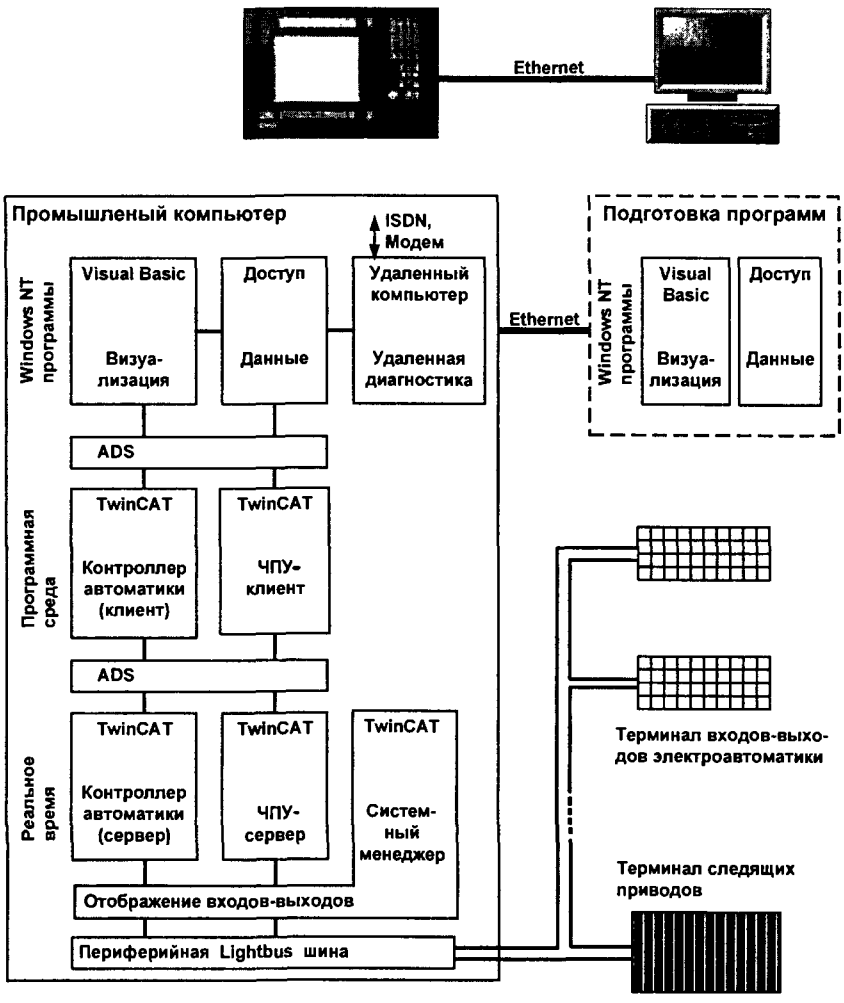


Рис. 6. Архитектура системы ЧПУ класса PCNC-4 фирмы Beckhoff

Внешний интерфейс выстроен на базе любой стандартной (по выбору) периферийной шины Fieldbus, в частности на базе шины Lightbus фирмы Beckhoff. Эта шина выполнена в виде кольцевого канала для передачи сигналов управления автономными следящими приводами, а также сигналов электроавтоматики. Выход к объектам осуществляется с помощью периферийных терминалов ввода-вывода. Операционная среда представляет собой комбинацию Windows NT для поддержания процессов машинного времени и системы TwinCat (Total Windows Control and Automation Technology).

Операционная система TwinCat фирмы Beckhoff интегрирована в Windows NT, добавляет ей функции реального времени, не изменяя самой Windows NT. Перемещение данных и доступ к прикладным функциям API программных модулей осуществляется через программную шину ADS (Automation Device Specification).

Системный менеджер, являющийся подсистемой TwinCat, служит центром системной конфигурации, поддерживающим синхронное или асинхронное взаимодействие всех процессов, а также ввод-вывод сигналов управления. На прикладном уровне в потоках управления работают программные модули ЧПУ и программируемые контроллеры, имеющие клиентскую (для подготовки данных) и серверную (для работы в реальном времени) части. ЧПУ-клиент интерпретирует кадры управляющей программы в стандарте DIN 66025, а ЧПУ-сервер выполняет интерполяцию в группах приводов – по три координаты в группе. Группы формируются системным менеджером. Для безэквидистантных программ можно обойтись без интерпретации, которую заменяет компилятор клиента контроллера автоматики. Одновременно работают до четырех контроллеров (виртуальных процессоров, выполненных в стандарте IEC 1131-3), каждый из которых решает четыре задачи, имеющих свой приоритет и свое время цикла.

Система ЧПУ фирмы Power Automation (Германия) построена на основе промышленного персонального компьютера с PCI-шиной (рис. 7), операционной системой Windows NT и ядром реального времени (собственной разработки). Операционная система Windows NT поддерживает работу интерфейса оператора, в том числе системы программирования ЧПУ и контроллера электроавтоматики, встроенную CAM-систему (опирающуюся на базы данных инструментов, материалов и технологических циклов), приложения конечного пользователя. Ядро реального времени синхронизирует задачи ЧПУ с электроавтоматикой, диспетчеризует работу интерпретатора, интерполятора и модуля управления следящими приводами. Одновременно могут работать до восьми каналов ЧПУ и два программно-реализованных контроллера электроавтоматики с разными приоритетами.

Система имеет открытую архитектуру, которая допускает расширение функций ядра ЧПУ за счет специальных функций пользователя (compile cycles – терминология Power Automation) (рис. 8; см. также описание системы Siemens), и исключительно мощное сетевое окружение (рис. 9), как внешнее (Ethernet-TCP/IP, Novell), так и периферийное (восемь оптоволоконных SERCOS-колец для 64 следящих приводов, InterBus-S, Profibus DP, CAN-Bus, ASI-BUS). Кроме того, предусмотрены собственная периферийная SUPERBUS-шина для удаленных вхо-

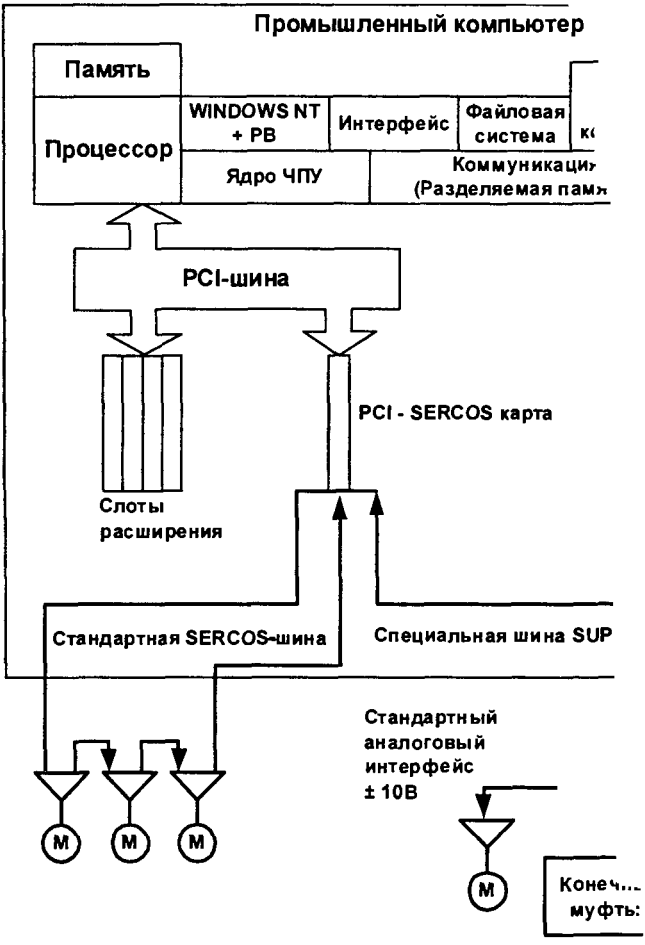


Рис. 7. Архитектура системы ЧПУ класса PCNC-4 фирмы ...
PB – ядро реального времени

дов-выходов электроавтоматики, а также удаленные входы-...
дарта фирмы OMRON (Япония) на PCI-шине. На SUPERK...
могут быть установлены четыре интерфейсных платы для ...
32 аналоговыми приводами.

Фирма Siemens не раскрывает особенностей своей архитек...
этом плане можно лишь строить догадки. Однако обращает на се...
ние механизм поддержания открытой архитектуры, которым обо...
идентичен такому же механизму фирмы Power Automation. В этой с...

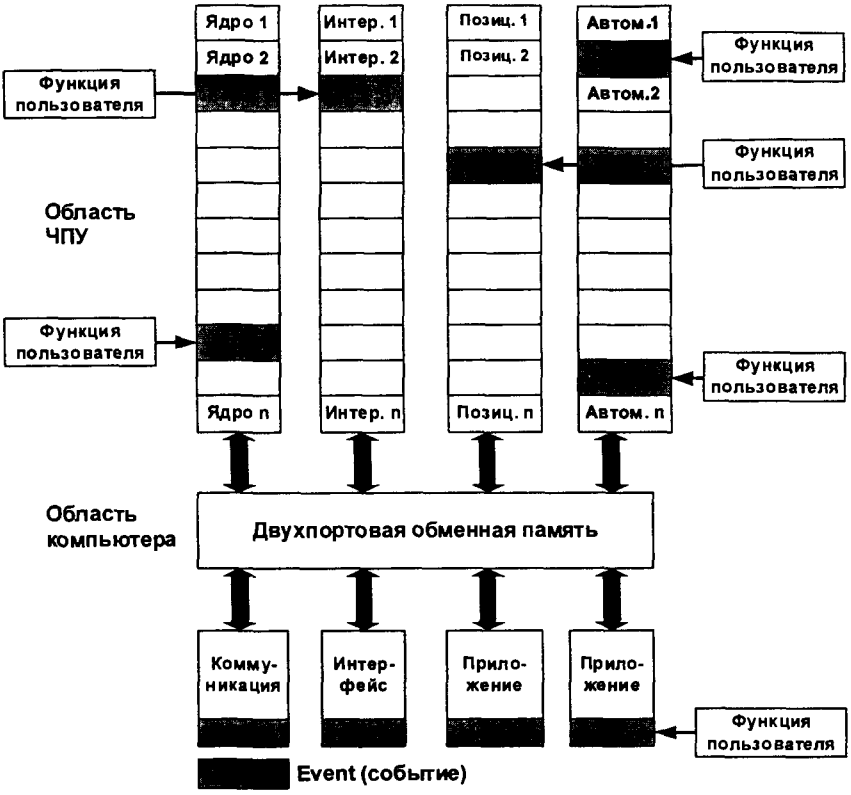


Рис. 8. Схема расширения функций ядра ЧПУ в системе фирмы PowerAutomation:
Ядро – модули ядра ЧПУ; Интер. – модули интерполяции;
Позиц. – модули связи со следящими приводами подачи;
Автом. – модули управления электроавтоматикой

по-видимому, можно предположить и идентичность архитектур Siemens и Power Automation.

При разработке новой модели системы ЧПУ фирма Siemens сделала акцент на открытую для конечного пользователя архитектуру со стороны как интерфейса оператора, так и ядра системы (рис. 10). Интерфейс оператора работает в операционной системе Windows NT, поэтому включение в интерфейс windows-приложений проблемы не составляет. Однако возможна и сравнительно глубокая реконфигурация интерфейса с помощью оригинальной инструментальной системы ProTool.

Для расширения функций ядра предложена схема, в соответствии с которой в процессы ядра включены своеобразные точки останова «break-out-points», называемые «events» (события). События инициируют фраг-

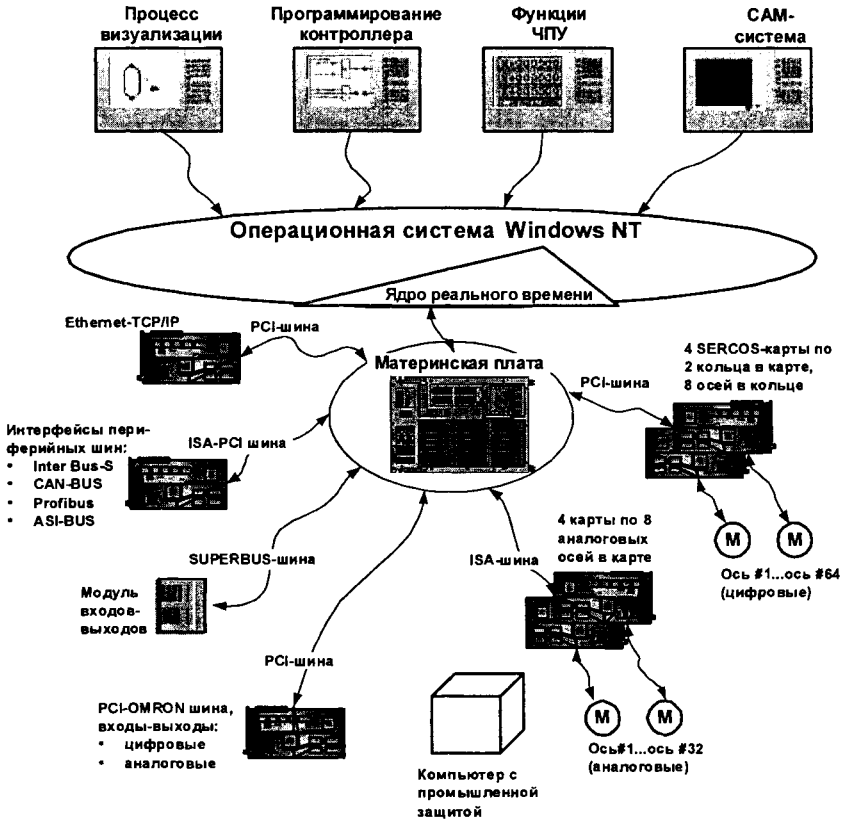


Рис. 9. Сетевое окружение системы ЧПУ фирмы Power Automation

менты пользовательского кода на Visual C++, называемого здесь «compile cycles» (скомпилированные циклы). Скомпилированные циклы имеют унифицированный интерфейс OPI (OEM Program Interface), что обеспечивает им стандартный доступ к системным данным и функциям посредством механизма связывания «binding». С другой стороны, скомпилированные циклы могут использовать и собственные данные. Такой подход обеспечивает полную совместимость расширенного программного обеспечения системы ЧПУ.

Для сравнения обратимся вновь к архитектуре системы ЧПУ фирмы Power Automation. Даже беглое сопоставление и одинаковая терминология указывают на то, что в обоих случаях использованы одинаковые механизмы внедрения функций конечного пользователя, т.е. одинаковый подход к реализации открытой архитектуры.

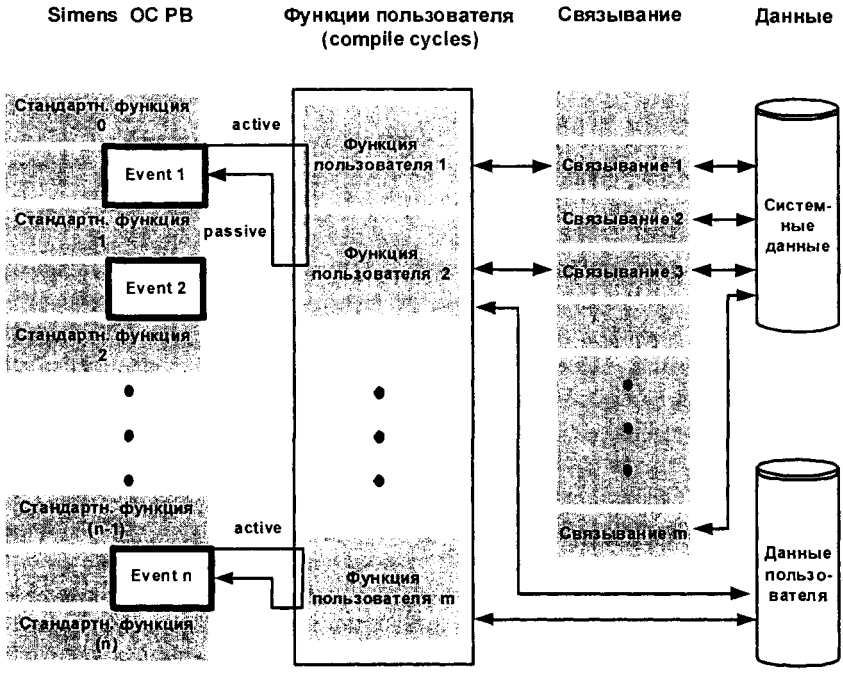


Рис. 10. Механизм поддержания открытой архитектуры в системе ЧПУ фирмы Siemens

Заключение

В спектре архитектурных решений наиболее уверенные позиции занимает концепция PCNC, при этом по мере роста вычислительной мощности процессоров все чаще предпочтение отдают однокомпьютерному варианту. В качестве операционной системы стандартом де-факто стала Windows NT с расширением реального времени [4]. Программируемые контроллеры реализуют программным путем в рамках единой вычислительной среды для ядра ЧПУ, а терминал системы ЧПУ используют для программирования электроавтоматики.

Периферия систем ЧПУ становится сетевой, причем все чаще единая сеть используется как для приводов подачи, так и для системы управления электроавтоматикой. Наиболее значительная тенденция состоит в развитии идей открытой архитектуры [5], предоставляющей конечному пользователю широкие возможности для реализации собственных функций.

1.2. Интеграция на основе открытого управления и стандарта OPC

Первоначально интерфейсный стандарт OPC был разработан для единообразия связи различных логических устройств, источников данных и периферийных сетей (Fieldbus) с клиентскими приложениями. Успех внедрения стандарта привел к расширению сферы его применения, и в эту сферу попали устройства ЧПУ, системы SCADA, терминальные системы интегрированной технологической среды. Поскольку стандарт OPC оказался в области интересов разработчиков систем ЧПУ, далее рассмотрены проблемы открытого управления на основе этого стандарта.

Еще недавно казалось, что проблема открытого управления в гетерогенных интегрированных системах решена на основе стандартов архитектуры открытых систем ISO-OSI и концепции MAP (Manufacturing Automation Protocol) [7, 8]. Однако практика построения интегрированных систем показала, что реализация MAP дорога, громоздка и не охватывает всех иерархических уровней интеграции, например нижних уровней для устройств типа программируемых контроллеров, следящих приводов и др. Система MAP скорее ориентирована на однородную архитектуру, тогда как современные решения все чаще строят на базе клиент-серверной организации, в которой заметную роль играют системы типа SCADA (Supervisory Control and Data Acquisition). В этой связи в последнее время формируется комплекс новых идей, которые переносят акцент с открытой архитектуры локальных систем (эта проблема во многом уже решена) на открытое управление в распределенных системах. В этом плане все возрастающий интерес приобретает промышленный стандарт OPC, который является составной частью COM-технологий фирмы Microsoft. В дальнейшем рассмотрен круг вопросов, относящихся к построению открытых интегрированных систем управления.

1.2.1. Представление об открытом управлении

Обратимся прежде к канонической архитектуре ISO-OSI (рис. 11, *а*). Стандартное приложение MMS (Manufacturing Message Specification, ISO 1090), работающее на прикладном уровне ISO-OSI, превращает распределенную систему управления в совокупность взаимодействующих виртуальных устройств VMD (Virtual Manufacturing Device). Однако переусложненный стек протоколов VMD (все остальные уровни ISO-OSI) заставляет обратиться к более простым сетям TCP/IP и искать возможность их сосуществования с MMS.

На рис. 11, *б* показано решение, использующее RFC 1006 (Remote Function Call) в качестве эмулятора ISO-OSI услуг над протоколами TCP/IP.

Рис. 11. Каноническая модель ISO-OSI и ее эволюция

На рис. 11, *в* система MMS-услуг выстроена непосредственно над сокетами TCP/IP. Переходный вариант показан на рис. 11, *г*, где использована система RPC (Remote Procedure Call), а сравнительно более устойчивый вариант – на рис. 11,*д*, для которого протоколы TCP/IP в принципе необязательны. Открытое управление предполагает максимальное использование стандартов, не только сетевых, но и на уровне каждой отдельной системы управления.

Структура системы ЧПУ на рис. 12 представлена с позиций использования существующих стандартов. Роль интерфейсных стандартов исключительно велика, поскольку именно они создают основу для построения открытых распределенных систем управления. В особенности хотелось бы обратить внимание на эффективность интерфейсов на основе объектно-ориентированного подхода и таких Microsoft-технологий, как COM/DCOM/OLE/OPC [9]. Пример системы ЧПУ с объектно-ориентированными интерфейсами приведен на рис. 13.

Стандарт OPC представляет собой технологию OLE для управления технологическими процессами [10, 11]. Это стандартный метод для доступа к периферийным устройствам, системам SCADA или другим промышленным приложениям реального времени. OPC является спецификацией, или набором правил и процедур, разработанных с той целью, чтобы разнообразные приложения могли поддерживать диалог между собой. Цель стандарта – обеспечить совместную работу и взаимозаменяемость промышленных устройств от разных производителей. Имея утвержденный в стандарте набор интерфейсов, конечный пользователь может организовать взаимодействие и обмен данными между любыми распределенными компонентами системы.

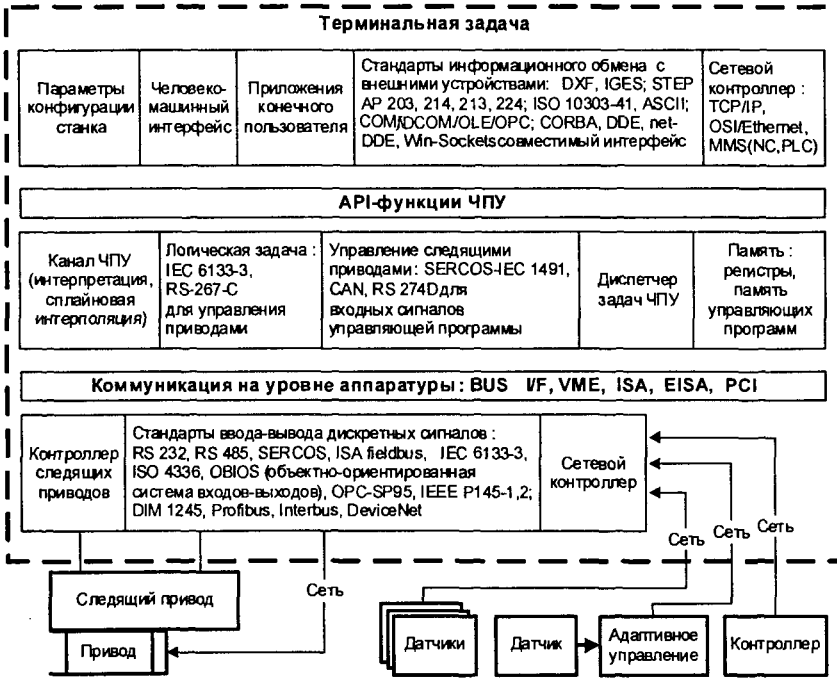


Рис. 12. Стандарты в современной системе ЧПУ

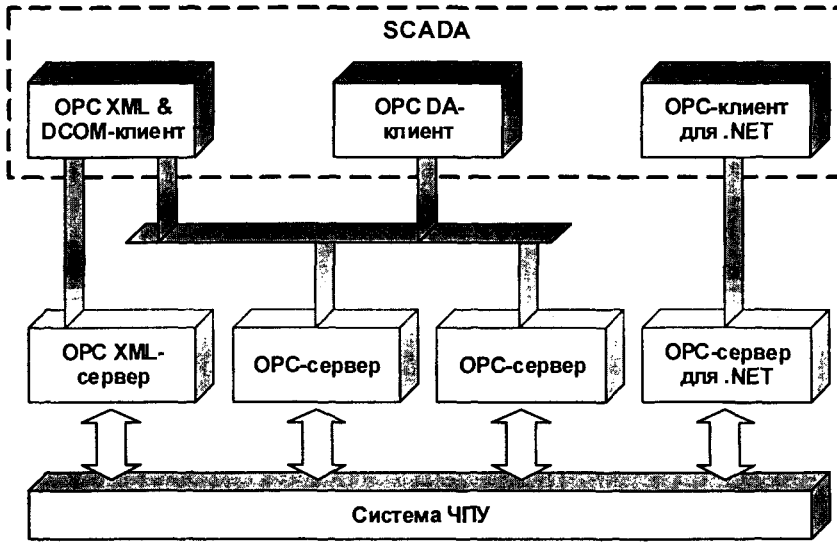


Рис. 13. Пример системы ЧПУ с объектно-ориентированными интерфейсами

Следует заметить, что в узлах современной гетерогенной системы находятся различные по назначению устройства (ЧПУ и др.), но построенные на аппаратуре персонального компьютера и на базе операционной системы Windows NT (в ближайшей перспективе это относится даже к программируемым контроллерам). Указанное обстоятельство существенно упрощает внедрение стандарта OPC.

На рис. 13 система ЧПУ (с совокупностью OPC-серверов) поставляет данные, а OPC-клиенты эти данные потребляют. К клиентам относятся устройства с программным обеспечением более высокого уровня, например системы SCADA. Такие системы в интегрированных системах могут быть потребителями данных ЧПУ, осуществляющими доступ к ним DA (Data Access). OPC-клиент предварительно запрашивает, может ли он работать с нужным ему интерфейсом OPC-сервера. В одной из версий DA-механизм уведомления клиента сведен к стандартному механизму COM/DCOM, что и показано на рис. 13.

Еще одна разновидность OPC-сервера – это доступ к периферийной шине (Fieldbus) устройства ЧПУ. В системе ЧПУ с операционной системой Windows устанавливается Fieldbus-адаптер, и OPC-сервер будет работать с сетью Fieldbus через драйвер адаптера. Таким образом, OPC-клиент типа .NET получает доступ к данным сети Fieldbus через OPC-сервер типа . NET.

С появлением стандарта OPC построение открытых распределенных систем управления становится достаточно простым еще и по той причине, что разработка OPC-серверов и OPC-клиентов поддержана сегодня многочисленными инструментальными средствами. С точки зрения SCADA-систем, OPC-серверы, расположенные на компьютерах всего производственного предприятия, могут стандартным способом поставлять данные в программы визуализации, базы данных и др. Таким образом, разрушается само понятие гетерогенной системы.

На рис. 14 OPC-интерфейсы системы ЧПУ представлены в явном виде. В качестве клиентов показаны системы MES (Manufacturing Execution Systems), ERP (Enterprise Resource Planning), MRP (Manufacturing resource Planning). Система MES отвечает за управление производственными ресурсами, планирование и контроль последовательности технологических операций, например в рамках гибкого производственного модуля или гибкой производственной системы. Система ERP занимается планированием ресурсов предприятия, а система MRP – планированием ресурсов производства в рамках подразделения предприятия.

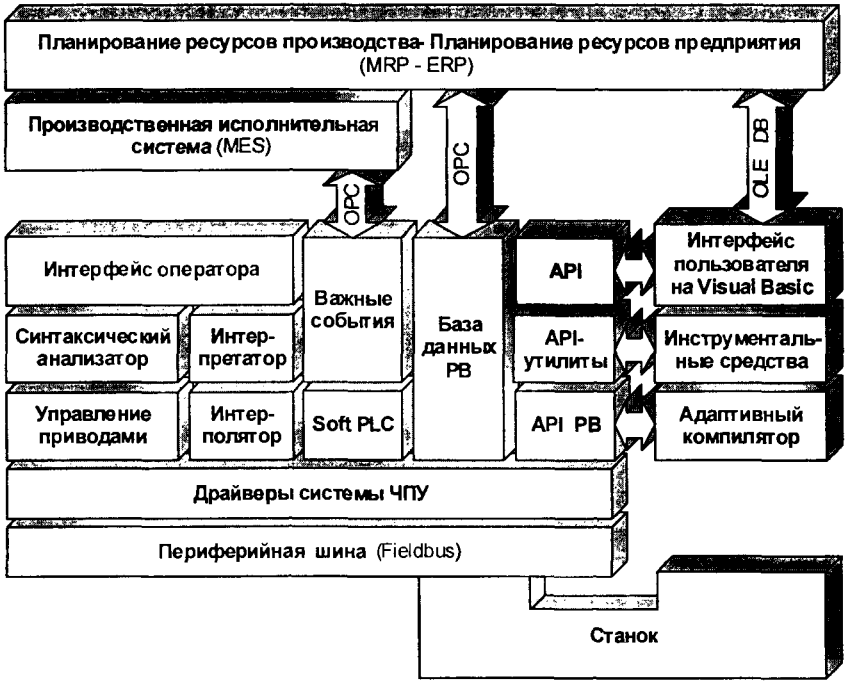


Рис. 14. Интерфейсы системы ЧПУ:
API (Application Programming Interface) – прикладной интерфейс; DB (Data Base) – база данных; Soft PLC – программно-реализованный контроллер электроавтоматики; MES – Manufacturing Execution Systems; ERP – Enterprise Resource Planning; MRP – Manufacturing resource Planning

1.2.2. Системы SCADA

Системы SCADA являются неизменными компонентами автоматизированной интегрированной системы. Они выполняют функции серверов технологических данных, поддерживающих обмен информацией между технологическими устройствами и сетью персональных компьютеров предприятия. В этой связи они могут выступать в роли терминальных станций и систем управления более высокого уровня, чем системы ЧПУ и программируемые контроллеры. В отдельных случаях системы SCADA могут выступать в роли терминала самой системы ЧПУ. Безотносительно к конкретному назначению функции систем SCADA формулируют следующим образом.

1. Сбор, первичная обработка и накопление информации о параметрах технологического процесса и состоянии оборудования от систем ЧПУ

и программируемых контроллеров, непосредственно связанных с технологическими машинами.

2. Отображение информации о текущих параметрах технологического процесса на экране монитора в виде графических мнемосхем.

3. Отображение графиков текущих значений технологических параметров в реальном времени за заданный интервал.

4. Обнаружение критических (аварийных) ситуаций. Вывод на экран монитора технологических и аварийных сообщений.

5. Архивирование истории изменения параметров технологического процесса.

6. Оперативное управление технологическим процессом. Диспетчирование устройств низшего ранга.

7. Предоставление данных о параметрах технологического процесса для их использования в системе управления предприятием.

Системы SCADA реализованы обычно в виде сетевых персональных компьютеров, причем необязательно все функции SCADA сосредоточены в одном компьютере. Так, в интегрированной системе могут быть выделены системы SCADA типов Data Access (доступ к данным технологического процесса), Alarms and Events (выявление критических и аварийных ситуаций), History Access (архивирование истории изменения параметров технологического процесса) (рис. 15).

Система Data Access считывает технологические параметры, сохраняет эти параметры в базе данных реального времени, отображает технологические параметры на графических мнемосхемах и в виде графиков (трендов). Система Alarms and Events обнаруживает аварийные ситуации, ото-

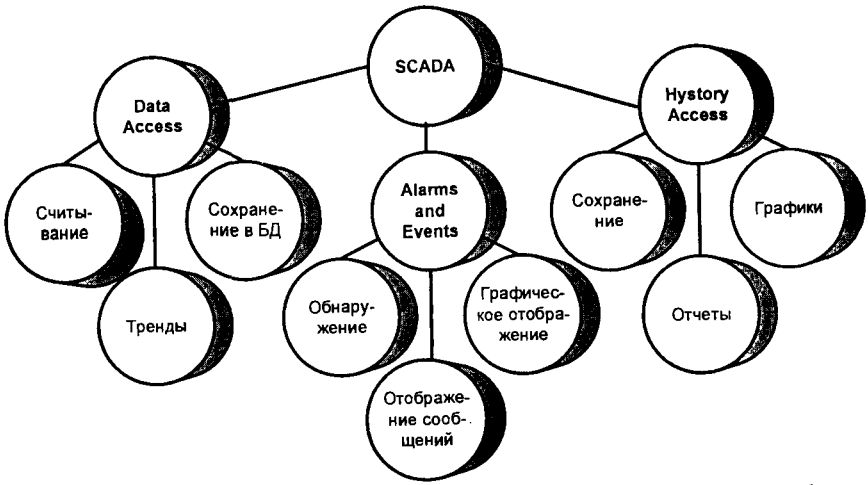


Рис. 15. Структура функционального назначения систем SCADA

бражает аварийные и технологические сообщения, динамически представляет аварийные ситуации на графических мнемосхемах. Система History Access архивирует историю изменения параметров технологического процесса, просматривает историю изменения параметров технологического процесса в виде графиков и таблиц, генерирует отчеты по истории изменения параметров технологического процесса.

Многие фирмы, производители систем SCADA, стараются сосредоточить в этих системах целый комплекс продуктов, удовлетворяющих всем потребностям автоматизации современного промышленного предприятия. Так, фирма Wonderware (США) выпустила продукт FactorySuit, в котором помимо стандартных функций SCADA реализованы следующие возможности: управление технологическими маршрутами (Batch Control), программирование контроллеров, ведение проектов, контроль качества продукции, некоторые функции автоматизации административного управления. Сегодня существует множество подобных примеров.

1.2.3. Стандарт OPC

Существует достаточно широкий набор интерфейсных OPC-стандартов: общие для всех OPC-спецификаций, для обмена оперативными данными с приложениями на C++ и Visual Basic, для обслуживания событий (event) и нештатных ситуаций (alarm), для работы с базами данными, для обработки прав доступа к данным и др. Основной стандарт, называемый DA (Data Access), описывает передачу оперативных данных от оборудования или к оборудованию. OPC-клиент может взаимодействовать с OPC-серверами от одного или нескольких производителей (рис. 16).

OPC Data Access-сервер состоит из нескольких объектов: сервера, группы, элемента данных (переменной). Объект-сервер поддерживает информацию о сервере и служит контейнером для объектов-групп. Объект-группа поддерживает информацию о самом себе и предоставляет механизм для включения и логической организации объектов-элементов. OPC-группы создают клиентам возможность организовывать данные. Например, группа может выводить элементы на экран монитора оператора или представлять их в сообщении; группы могут обслуживать разных клиентов. Данные можно читать и писать. OPC-клиент может сконфигурировать скорость, с которой OPC-сервер будет обновлять его данные.

Существуют два типа групп: public и local (или private). Тип public служит для разделения групп между многими клиентами, тип local предназначен для одного клиента. В пределах группы клиент определяет один или более OPC-элементов (рис. 17).

OPC-элементы устанавливают связи с источниками данных в пределах сервера. С позиций специального интерфейса OPC-элемент недоступен для

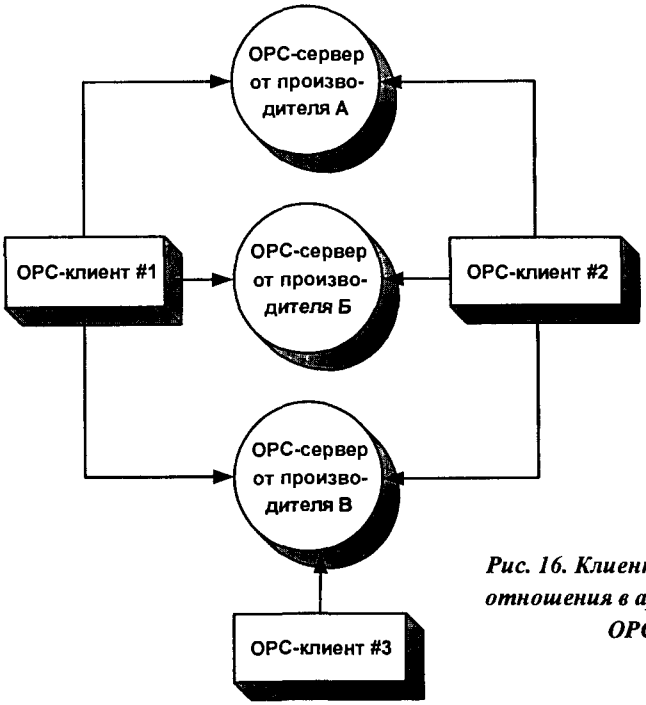


Рис. 16. Клиент-серверные отношения в архитектуре OPC

OPC-клиента как объект. Другими словами, не существует внешнего интерфейса, который был бы определен для OPC-элемента. Все виды доступа к OPC элементам осуществляются посредством OPC объектов-групп, которые содержат OPC-элементы. Элементы-переменные не служат источниками данных, они представляют собой лишь соединения с ними. OPC-элемент следует рассматривать как нечто, специфицирующее адрес данных, а не физический источник данных, на который адрес ссылается.

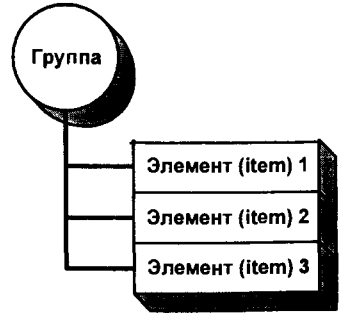


Рис. 17. Отношение между группой и элементами

Таким образом, основной единицей данных в OPC служит переменная (item). Переменная может быть любого типа, допустимого в OLE: различные целые и вещественные типы, логический тип, строковый тип, дата, вариантный тип и др. Кроме того, переменная может быть массивом. К обязательным свойствам переменной относятся значение Value и тип, качество переменной Quality,

метка времени Time Stamp, права доступа (чтение, запись), частота опроса OPC-сервером и описание переменной.

Качество предполагает, что в источниках данных возможны отказы, поэтому корректное значение переменной не всегда известно OPC-серверу, о чем и уведомляется клиент через качество – хорошее, плохое, неопределенное, дополнительная информация. Метка времени сообщает, когда переменная получила конкретное значение и качество. Частота опроса определяет интервал чтения переменной. Описание переменной представляет собой строковое значение, содержащее информацию для пользователя о предназначении переменной.

Объекты OPC-сервера напоминают обычные COM-объекты (рис. 18).

Существуют три способа получения OPC-клиентом данных от OPC-сервера: синхронное чтение, асинхронное чтение и подписка. При синхронном чтении клиент посылает серверу запрос со списком интересующих его переменных и ждет, пока сервер его выполнит. При асинхронном чтении клиент посылает серверу запрос, а сам продолжает работать. Когда сервер выполнил запрос, клиент получает уведомление. В случае подписки клиент передает серверу список интересующих его переменных, а сервер затем регулярно присылает клиенту информацию об изменениях значений переменных из списка. Эти списки в терминологии OPC называют группами. Каждый клиент может поддерживать одновременно много групп с разной скоростью обновления.

Технология OPC регламентирует только интерфейс между OPC-клиентами и OPC-серверами, но не устанавливает способ получения этих данных от оборудования. Однако существуют некоторые модели взаимодействия с оборудованием, предполагаемые разумными с точки зрения разработчиков OPC. Например, можно запросить OPC-сервер получать данные не напрямую, а извлекать их из своего внутреннего буфера (кэша).

Переменные в OPC-сервере могут быть упорядочены в простой список или в «дерево», напоминающее «дерево» файлов на диске. Имеются соответствующие интерфейсы для навигации по этому дереву. Можно, в частности, в любой момент запросить «дерево» переменных, поддерживаемых OPC-сервером. Есть механизм оповещения завершения работы OPC-сервера, возможность запроса информации о самом сервере и списка зарегистрированных групп.

Соответствующие интерфейсы предлагают OPC-клиентам некоторые механизмы, которые уведомляют о возникновении специфицированных событий или аварийных ситуаций. Они оказывают OPC-клиентам услуги, позволяющие идентифицировать события и условия, поддерживаемые OPC-сервером, а также получать текущий статус.

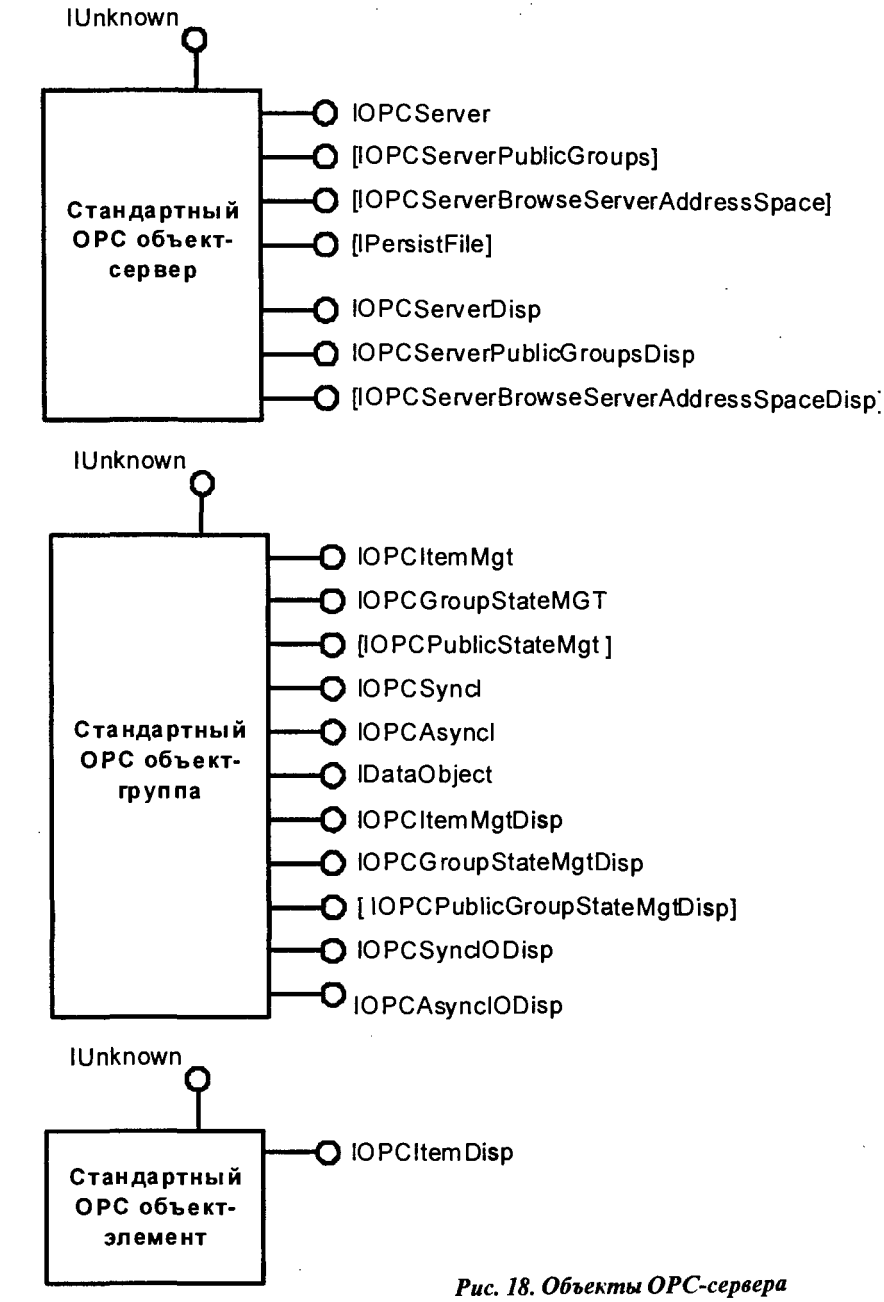


Рис. 18. Объекты ОРС-сервера

Помимо серверов, осуществляющих доступ к данным DA, существуют серверы доставки сообщений типа «alarms» и «events». В OPC alarm определяют как нерегулярную ситуацию, представляющую собой особый случай условий (condition). Условие – это поименованное состояние событийного OPC-сервера или одного из его вложенных объектов. Примерами условий могут служить, например, HighAlarm, HighHighAlarm, Normal, LowAlarm, и LowLowAlarm.

С другой стороны, событие (event) является различимой ситуацией, которая имеет значение для OPC-сервера, представляемого им устройства и заинтересованных OPC-клиентов. Событие может быть, а может и не быть ассоциировано с условием. Например, переходы из HighAlarm в Normal есть события, ассоциированные с условиями. Интерфейс OPC-сервера предлагает методы, позволяющие OPC-клиенту: устанавливать типы событий, которые поддерживает OPC-сервер; подписываться на специфические события с тем, чтобы OPC-клиенты могли получать уведомления об этих событиях; осуществлять доступ к условиям и управлять условиями, создаваемыми OPC-сервером.

Еще одна категория OPC-серверов – это так называемые «машины регистраций», которые создают дополнительный источник информации, распространяемый между заинтересованными клиентами. Обзорную информацию можно посчитать особым типом данных. Существуют несколько типов серверов просмотра данных, основные из которых – простые серверы трендов данных, а также серверы, осуществляющие компрессию и анализ сложных данных.

Спецификации OPC всегда содержат два набора интерфейсов: интерфейсы пользователя (Custom Interfaces) и интерфейсы автоматизации (Automation Interfaces). Последние имеют доступ к приложениям, написанным на Visual Basic (рис. 19). Спецификации OPC определяют, что собой представляют интерфейсы, но не как выглядит проект. Специфицируется ожидаемое поведение интерфейсов, которое позволяет клиентским приложениям их использовать. Представлены описания архитектурных

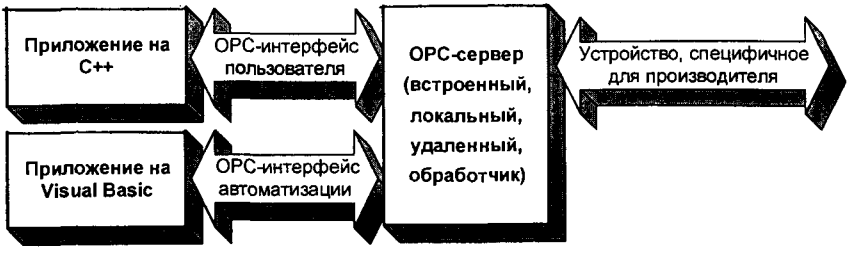


Рис. 19. OPC-интерфейсы

решений и интерфейсов, которые наиболее подходят для архитектурных решений. При разработке OPC-сервера принимают ряд важных решений: выбор частоты передачи данных по выделенным каналам к физическим устройствам или другим источникам данных, выбор локального или удаленного устройства, располагающего кодами, ответственными за сбор данных.

Клиентское OPC-приложение взаимодействует с OPC-сервером через специфицированные разделяемые интерфейсы, специальный (пользовательский) интерфейс или интерфейс автоматизации. Прежде всего OPC-серверы нуждаются в разработке специальных (пользовательских) интерфейсов, а в качестве опции могут иметь интерфейсы автоматизации. В некоторых случаях интерфейс автоматизации располагает специальной DLL-оболочкой, которая может быть использована для его построения на основе интерфейса пользователя (рис. 20).

Рис. 20. Типичная архитектура OPC

Заключение

Как только предполагается, что система ЧПУ является компонентом интегрированной технологической среды, возникает проблема открытого управления, означающего совместимость и интерактивность разнообразных локальных устройств. Проблема открытого управления состоит в максимальном использовании стандартов на всех уровнях и в первую очередь – интерфейсных стандартов. Появление таких стандартов существенно расширило арсенал технических средств интегрированной технологической среды за счет систем типа SCADA.

Системы SCADA, локальные системы управления, терминальные станции вступают в клиент-серверные отношения, причем интерфейсы клиентских и серверных приложений взаимодействуют через OPC-интерфейсы. Интерфейс OPC перспективен, и производителям систем ЧПУ следует

иметь в виду его абсолютную необходимость, поскольку независимая и автономная работа систем ЧПУ уходит в прошлое.

1.3. Интеграция на основе комплекса производственных стандартов STEP (Standard for the Exchange of Product model data)

Стандарт STEP используют для создания информационной модели изделия, работающей на всех этапах его жизненного цикла. Этап перехода от системы автоматизированного программирования CAM к системе ЧПУ называют STEP-NC. От окончательного международного согласования этого этапа ожидают перехода к кардинально более совершенной системе программирования ЧПУ и изменений в самой архитектуре систем ЧПУ.

Среди возможных видов интеграции в автоматизированных производствах в последнее время привлекают те, которые построены на единой информационной модели изделия в рамках его жизненного цикла: от компьютерного проектирования (CAD) и компьютерного планирования (CAPP) к автоматизированной подготовке управляющих программ (CAM) и изготовлению на станках с ЧПУ (NC). Подобная модель определена в рамках комплекса стандартов STEP [12, 13]. Слабым звеном в последовательных переходах по этапам жизненного цикла является переход CAM-NC, уверенное представление о котором не сложилось до сих пор. В дальнейшем сделан акцент именно на этом переходе, что потребовало, однако, введения общих представлений о комплексе стандартов STEP.

1.3.1. Обзор комплекса производственных стандартов STEP

Речь далее пойдет о той части стандартов STEP, которая определена для области обработки резанием на станках с ЧПУ. В жизненном цикле изделия предусмотрены фазы: STEP-проектирования (CAD, Computer-Aided Design), макропланирования технологического процесса (CAPP, Computer-Aided Process Planning), а также те способы микропланирования (CAM, Computer-Aided Manufacturing) и изготовления (NC, Numerical Control), которые существуют сегодня вне STEP. Функции STEP ориентированного микропланирования и STEP ориентированного изготовления станут доступными в ближайшем будущем.

Фаза проектирования предполагает генерацию и сохранение STEP-данных для последующего производства изделий. В рамках фазы разработаны несколько вариантов прикладных протоколов (AP, Application Protocol), определенных в качестве международных стандартов, наилучшим из которых служит протокол AP224. На уровне этой фазы формируется некото-

рый полный набор информации для планирования технологических маршрутов в очередной фазе. Полнота означает определение данных в терминах 3D-геометрии (прямые, дуги и т.д.), но и в таких технологических терминах, как карманы, канавки, отверстия, скругления и др. Полнота означает также определение размеров и допусков, ассоциированных с 3D-образом, генерацию такой существенной информации, как материал, шероховатость, специальные технические требования (например, скругление острых кромок).

Все спецификации представляют собой не просто текст в виде примечаний, но являются частью модели, причем под протоколом AP224 понимают и модель, и транслятор, генерирующий производственные данные для отдельных деталей и сборок в формате AP224. В составе транслятора имеется система управления базой данных(СУБД). В фазе проектирования создают проект, выполненный в CAD-системе, или используют уже существующий проект, транслированный в AP224.

В следующей фазе макропланирования используют производственные данные конструкторского проекта и обрабатывают их соответственно новым задачам. Представление данных в формате AP224 существенно повышает эффективность планирования; окончательный же результат макропланирования будет представлен в формате AP213 в форме технологического маршрута для станков с ЧПУ. Формат AP213 принадлежит комплексу STEP, но пока еще не является международным стандартом, хотя имеет мощную международную поддержку. Пока же в основном STEP служит только входом в систему макропланирования, в то время как выход организован в формате используемой CAPP-системы.

Один из существующих вариантов CAPP-системы разработан в виде машины знаний, как интеллектуальное приложение для CAD-системы. Информация об изделии, о цеховых ресурсах, специфические сведения о построении технологических маршрутов и практическом опыте объединены вместе с целью построения планов обработки, используемых для самого широкого спектра деталей.

План обработки содержит схему маршрутизации (распределение шагов маршрута по станкам), спецификацию материалов, обобщающую маршрутную информацию, требования к инструментальному обеспечению, нормы времени для каждого шага, инструкции оператору. Приспособления и инструменты выбираются, заказываются или изготавливаются.

В процессе макропланирования оценивают стоимость обработки. Система принимает информацию в форматах STEP AP224 (оптимальный вариант), STEP AP203 (более ранний вариант прикладного протокола проектирования), IGES (Initial Graphics Exchange) и информацию чертежа. Модель цеховых ресурсов включает наличные материалы и инструменты, описания станков, оценки времени обработки, технологические возмож-

ности. Пользователю доступна твердотельная модель обрабатываемого изделия. В его распоряжении имеется множество экранов с информацией об изделии, цеховых ресурсах и плане обработки.

Стратегия очередной фазы – микропланирования – состоит в том, чтобы принять информацию в формате AP213 на основе формата AP224, но это станет возможным, когда формат AP213 будет завершен и выстроен в качестве стабильной модели и стандартного входа в CAM-системы. CAM-система выполнит микропланирование в формате AP238 на основе стандарта STEP-NC для каждого станка из тех, которые определены маршрутом операций.

В любом случае выход STEP-ориентированного макроплана используют в качестве входа в систему микропланирования. Микроплан ориентирован на шаги операций, поддержанные числовым программным управлением; он содержит чертежи установок и управляющие программы для станков с ЧПУ. CAM-системы проектируют траектории инструментов и постпроцессируют их, чтобы обеспечить совместимость с конкретной системой ЧПУ. Кроме того, разрабатываются схемы установок и коррекции инструментов, а также подробные инструкции оператору.

Фаза изготовления деталей станет гораздо более совершенной после завершения и внедрения стандарта AP238 (STEP-NC). Однако для прямого использования инструкций STEP AP238 должны быть разработаны системы ЧПУ очередного поколения, такие, которые понимают формат STEP-NC вместо языка ISO-7bit (ISO 6983, DIN 66025).

1.3.2. STEP-NC

Программирование современных систем ЧПУ подчиняется стандарту ISO 6983 (DIN 66025), которому уже более 50 лет и который явно тормозит развитие ЧПУ-технологии (рис. 21). Стандарт поддерживает простые команды для элементарных перемещений и логических операций. Управляющие программы в стандарте ISO 6983 содержат ничтожное подмножество информации, полученной на уровне систем CAD-CAM. Однако более серьезным является невозможность двустороннего обмена информацией с этими системами. Это означает, что любые изменения в управляющей программе не могут быть отображены в восходящем информационном потоке к системам CAD-CAM.

В отличие от существующего, стандарт STEP-NC предлагает модель того, *что* нужно сделать, но не подробности того, *как* осуществлять траекторные перемещения и выполнять команды логических переключений. Эта модель отвечает новому стандарту ISO 14649, согласно которому изделие получают из заготовки путем удаления типовых форм (*features*), условного или безусловного выполнения ассоциированных с типовыми формами переходов (*workingsteps*), в потоке управления, задаваемом исполняемы-

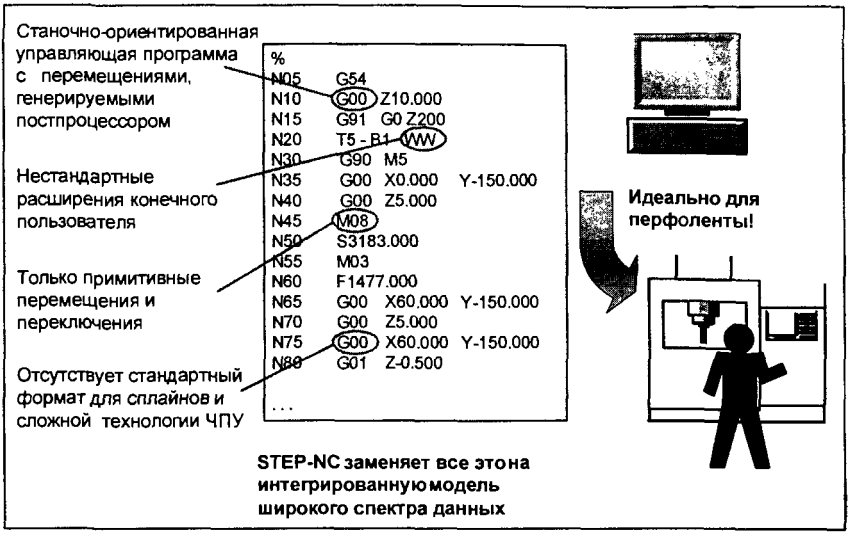


Рис. 21. Существующая схема программирования станков с ЧПУ и ее недостатки

ми блоками (*executables*), с необходимыми допусками, с использованием инструмента, отвечающего всем необходимым требованиям. Эта модель использует информацию форматов AP204 и AP213 вплоть до этапа интерпретации управляющей программы, т.е. она несопоставимо богаче существующей схемы программирования. Предполагается, что система управления способна интерпретировать подобную информацию и генерировать необходимые перемещения и циклы.

Стандарт ISO 14649 устанавливает девять компонентов функциональности (Units of Functionality, UOFs): проект (*project*), изделие (*workpiece*), типовую форму (*feature*), исполняемый блок (*executable*), переход (operation), траекторию инструмента (*toolpath*), измерения (*measures*). Отношения между компонентами показаны на рис. 22.

Рисунок представлен в форме, соответствующей упрощенной графической версии объектно-ориентированного языка EXPRESS [14, 15], который послужил средством описания всех прикладных протоколов STEP.

Изделие описывают так, как это принято в стандарте STEP: с историей версии, информацией владельца, утверждениями, датой, указанием материала и его свойств. Изделие служит выходом технологического процесса, а его внешний вид является свойством готового продукта. Типовые формы определяют области удаляемого материала заготовки, а их внешний вид является частью внешнего вида изделия. Типовые формы задают в параметрическом виде как совокупность образующей и направляющей. Осо-

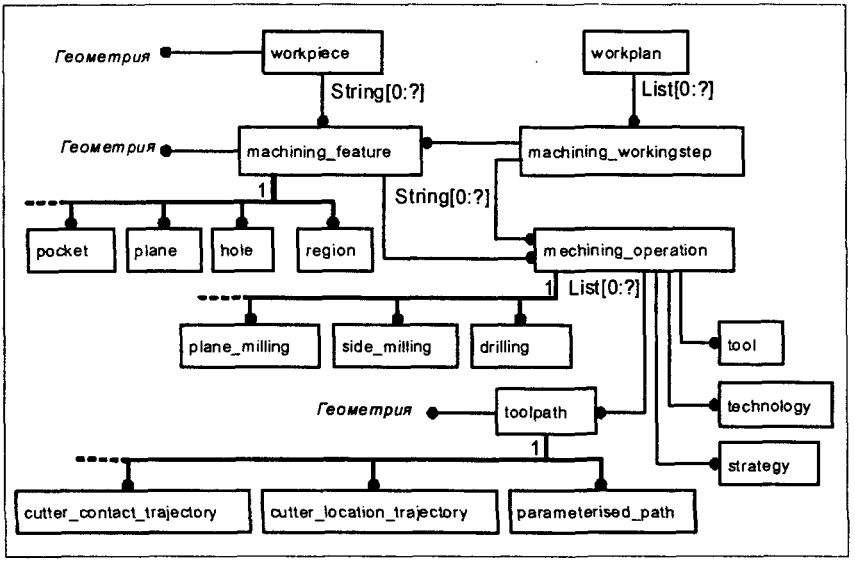


Рис. 22. Отношения между компонентами функциональности в стандарте ISO 14649

бый случай представляют поверхности свободной формы, для которых задают область, в пределах которой они размещаются. Некоторые виды типовых форм представлены на рис. 23.

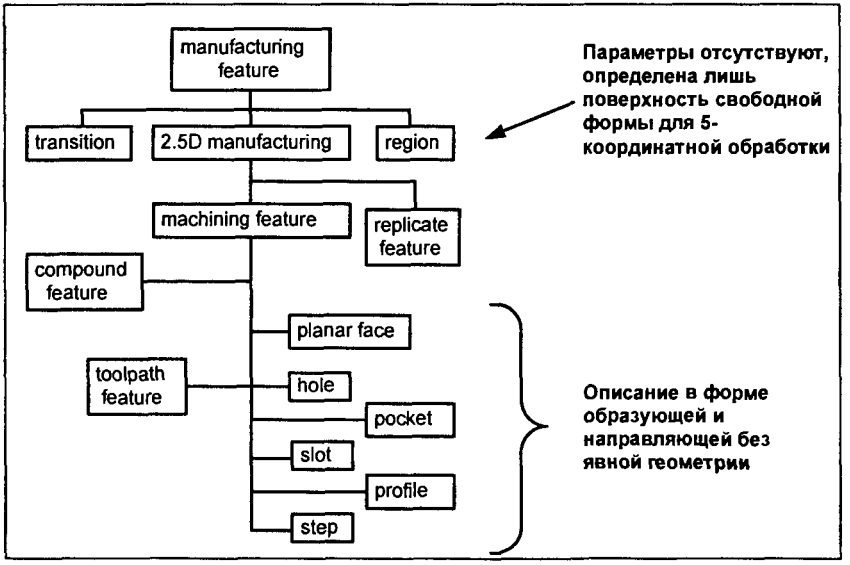


Рис. 23. Виды типовых форм

Ядро модели STEP-NC составляет план операций (*workplans*), который является последовательностью шагов операции (*workingsteps*). Каждый шаг операции ассоциирован с переходом, выполняемым в некоторой типовой форме изделия. В свою очередь переход содержит технологический алгоритм (включая стратегию внедрения в материал и вывода инструмента) и указания по настройкам. Переходы имеют черновую и чистовую версии. Предполагается, что интеллектуальные системы ЧПУ будут самостоятельно рассчитывать траектории инструмента для стандартных типовых форм.

Исполняемый блок (*executable*) описывает поток управления и последовательность переходов, ассоциированных с операциями и типовыми формами. Исполняемый блок технологически независим. Конструкция исполняемого блока приведена на рис. 24.

Траектория инструмента устанавливает точное движение координатных приводов в том случае, если интеллектуальная система ЧПУ неспособна сама спланировать такую траекторию. Однако полная траектория может быть воссоздана из каких-то ее повторяющихся или стандартных частей. Таким образом, гибкость плана операций снижается лишь частично. Структура траектории инструмента представлена на рис. 25.

Компонент функциональности *измерения* определяет используемые средства измерения и допуски в разработанной модели.

Обобщающим компонентом функциональности служит *проект*. Суть в том, что общая модель STEP-NC может включать описания нескольких изделий и

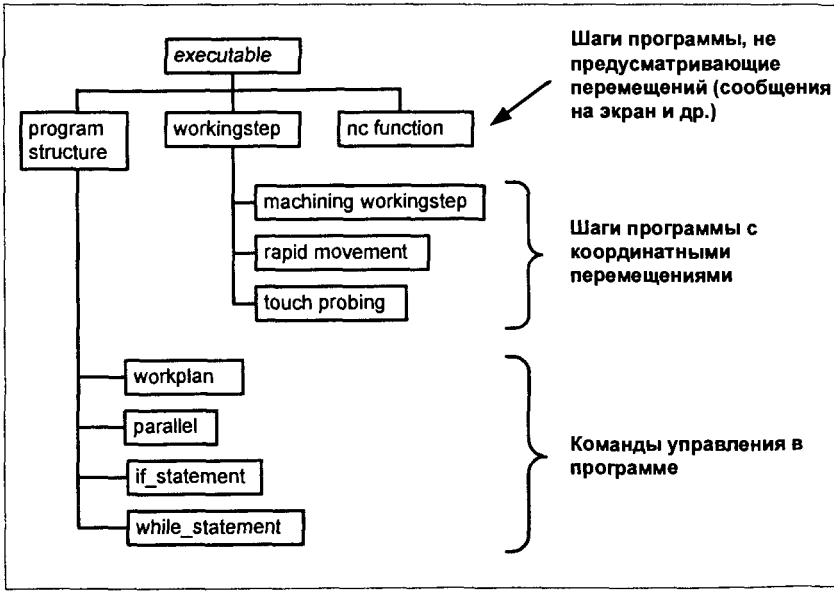


Рис. 24. Конструкция исполняемого блока

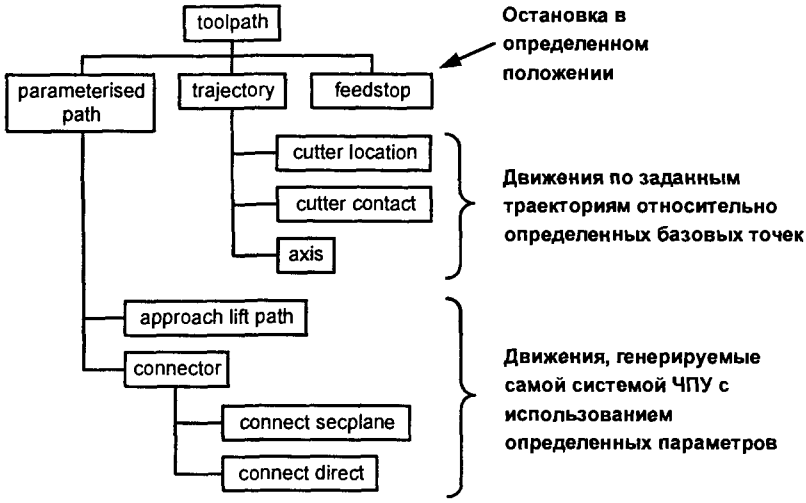


Рис. 25. Структура траектории инструмента

множество планов операций. Проект устанавливает стартовую точку, идентифицируя главный план операций. Формальное описание проекта в объектно-ориентированном языке EXPRESS выглядит следующим образом:

```
ENTITY project;
its_id: identifier;
main_workplan: workplan;
its_workpieces: SET [0:?] OF workpiece;
its_owner: OPTIONAL person_and_address;
its_release: OPTIONAL date_and_time;
its_status: OPTIONAL approval;
(*
Informal proposition:
its_id shall be unique within the part programme.
*)
END_ENTITY;
```

Управляющая программа для станка с ЧПУ представлена в формате физического файла, соответствующего ISO 1033, часть 21. Первая секция программы служит заголовком (*header*). Здесь представлена информация общего характера и комментарии (имя файла, автор, дата и др.) (рис. 26). Далее следует секция данных, открываемая ключевым словом Data. Эта секция делится на три части: план операций и исполняемые блоки, технологические описания, геометрические описания. Рисунок показывает отношения между этими тремя частями.

Рис. 26. Структура управляющей программы для станка с ЧПУ

План операции объединяет исполняемые блоки в линейном порядке или с учетом условий. Один из типов исполняемых блоков содержит структуру программы; чтобы изменить последовательность операций достаточно внести изменения в этот блок.

Пример использования управляющей программы в подобном виде был продемонстрирован фирмой Siemens. Далее приведен небольшой фрагмент такой программы:

```
File:
 Header
 #1=Project (Workplan #10);
 #10=Workplan (#20, #35, #71,...);
 ...
 #20=Machining_Workingstep (#(Feature), #22(Operation));
 #21=Round_hole ('Hole M6',,,,,,,);
 #22=Drilling (#...(Tool),,,#...(Technology), #...(Machine_functions));
 ...
 #35=Machining_Workingstep (...);
End-ISO-10303-21
```

Использование этого формата имеет ясное представление и четкое окружение, как это показано на рис. 27. Однако имеются и другие предложе-

Рис. 27. Окружение разработки управляющей программы в соответствии с ISO 10303-21

ния, связанные с прямым использованием в управляющих программах ЧПУ языков EXPRESS и XML. Дело в том, что синтаксис ISO 10303-21 не предполагает расширений и не предусматривает использование гипертекстовых механизмов.

1.3.3. Использование в интерфейсе систем ЧПУ языков EXPRESS и XML

Среди многих интересных достоинств такого подхода важную роль играет ориентация на интегрированное распределенное производство.

Язык EXPRESS является универсальным средством для описания информационных моделей в терминах «сущность – атрибуты». Сущности могут сохраняться в репозиториях в качестве абстрактных объектов, не имеющих привязки к конкретным физическим образам. Однако разработчики математического обеспечения репозиториев имеют возможность использовать любые информационные технологии и подходы при определении сущностей и атрибутов.

Язык XML гибок и расширяем, в этом смысле он имеет преимущества перед ISO 10303-21: XML-документы могут быть обработаны Web-браузерами, при этом технология браузеров позволяет визуализировать EXPRESS-сущности; XML-описания принимают участие в обмене нейтральными дан-

ными, но могут быть использованы в разделяемых специализированных базах данных и архивах [16].

Правила отображения и раннего связывания моделей EXPRESS и XML состоят в следующем: имя тэга соответствует или имени EXPRESS-сущности, или имени атрибута; элемент тэга – значениям атрибута; для упрощения структуры тэга его атрибуты являются многофункциональными. Упрощенная схема отображения моделей показана на рис. 28.

Окончательная схема генерации управляющей XML-программы ЧПУ из EXPRESS-модели данных ЧПУ показана на рис. 29. Эта модель охватывает EXPRESS-схему и EXPRESS-репозиторий. EXPRESS-схему можно

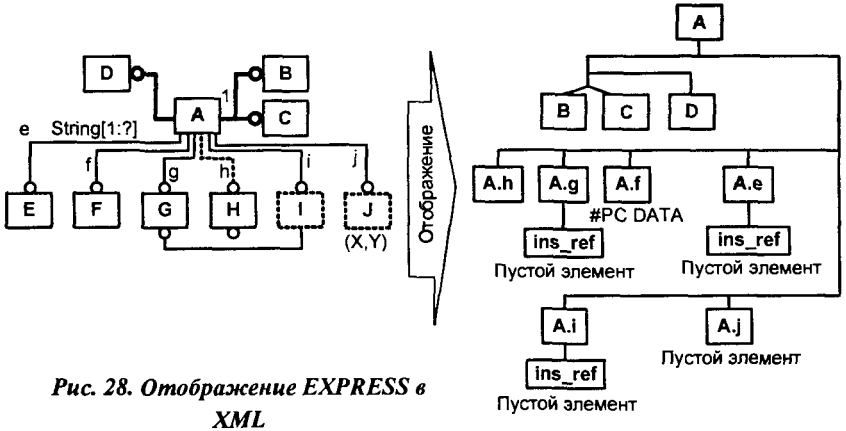


Рис. 28. Отображение EXPRESS в XML

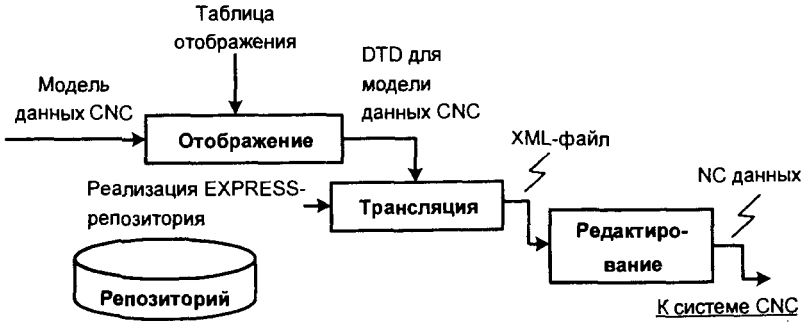


Рис. 29. Схема генерации управляющей XML-программы ЧПУ

конвертировать в XML DTD (Document Type Declaration) с использованием правил отображения.

Сопоставление моделей управляющих программ ЧПУ – современной (ISO 6983) и перспективной (ISO 14649) – выполнено на рис. 30.

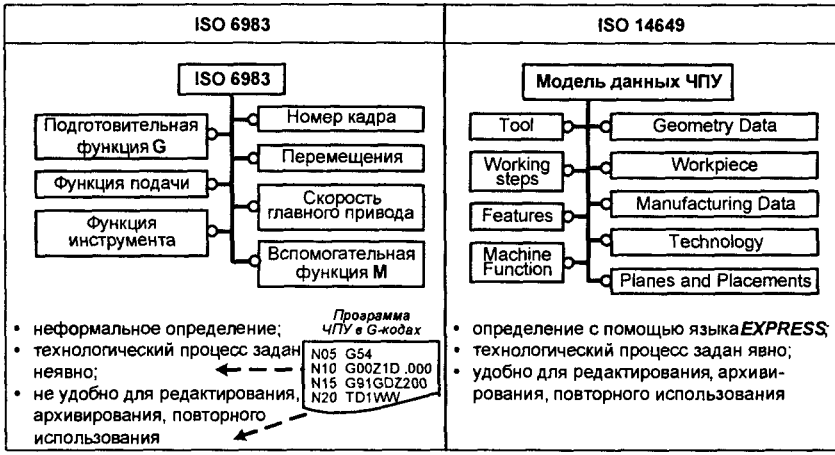


Рис. 30. Сравнение двух моделей управляющих программ ЧПУ: современной (ISO 6983) и перспективной (ISO 14649)

Заключение

До сих пор станки с ЧПУ программируют в стандарте ISO 6983. Этот стандарт существует со времени использования перфолент и перфокарт; он абсолютно не удовлетворяет современным технологиям. Управляющие программы, соответствующие ISO 6983, всего лишь описывают координатные перемещения (G1, G2, G3) и управляют циклами (M3, M8). Новые языки программирования работают с технологическими задачами, привязанными к типовым формам (features). Такой задачей может быть, к примеру, обработка кармана. Все операции, необходимые для перехода от заготовки к готовому изделию, могут быть описаны в терминах технологических задач. В этой связи на цеховой уровень поступает огромный объем информации. Все модификации цехового уровня могут быть не только сохранены, но и без труда переданы обратно в отделы планирования. Поскольку геометрия заготовки и готового изделия описывается с использованием STEP-синтаксиса, возможен прямой обмен информацией между CAD/CAM/CNC системами. Геометрические данные могут быть непосредственно импортированы в систему ЧПУ, при этом должна быть добавлена технологическая информация, чтобы сгенерировать управляющую программу.

Глава 2. Общие принципы построения систем ЧПУ

Концепцию системы ЧПУ разрабатывают аналитики, обладающие необходимым опытом и знаниями в области управления в реальном времени, обладающие способностью определять стратегию развития ЧПУ. При формировании концепции необходим список проблем, которые следует поставить и решить. В числе этих проблем: выбор архитектурного варианта, организация среды реального времени, выбор способа программирования и управления электроавтоматикой, формирование коммуникационной среды для осуществления транзакций между подсистемами и процессами.

2.1. Архитектура систем PCNC

Рассмотрены основные признаки систем ЧПУ нового поколения для мехатронных систем, в числе которых принадлежность к классу персональных систем управления PCNC и использование принципов открытой архитектуры. Отмечены достоинства открытой архитектуры двух- и однокомпьютерных систем: гибкость, клиент-серверная организация транзакций, объектно-ориентированный подход на уровнях макроструктуры и технологии программирования. Представлена организация системы ЧПУ, в которой модули с традиционными наименованиями имеют новое функциональное и алгоритмическое наполнение и новую программную реализацию. Указана особая роль PC-подсистемы, которая определяет пользовательские характеристики и уровень сервиса для оператора.

2.1.1. Признаки нового поколения систем ЧПУ

Очередная смена поколений существенно меняет потребительские свойства, структуру, архитектуру и математическое обеспечение систем ЧПУ. Огромный опыт, накопленный в области ЧПУ мехатронными системами, серьезно пересматривается под давлением производителей мехатронного оборудования и конечных его пользователей. В свою очередь производители систем ЧПУ прекрасно понимают, что простая эволюция традицион-

ных решений приведет к потере рынка и полному их забвению. Внешние причины подобной ситуации состоят в увеличении разнообразия мехатронных систем, ориентированных на решение специфических задач (разнообразные технологические машины, роботы, испытательные стенды и др.), расширение зоны активности оператора мехатронного оборудования, росте привлекательности персональных систем ЧПУ типа PCNC. Однако есть и глубинная внутренняя причина – внедрение новой объектно-ориентированной технологии, без которой создание мультимегабайтного программного обеспечения систем ЧПУ просто невозможно. Подобную технологию используют не только на уровне программирования (для повышения надежности и обозримости математического обеспечения), но и на уровне макропроектирования системы: основные модули определяют как «вложенные объекты», отношения между которыми носят клиент-серверный характер. Одним из вариантов общего решения является выделение глобального сервера – программной (виртуальной) шины, которая служит основным средством межмодульной коммуникации.

Принципиальной особенностью системы ЧПУ типа PCNC является использование открытой архитектуры, которая предполагает:

- конфигурирование системы у производителя мехатронного оборудования и конечного пользователя;
- интеграцию покупных программных пакетов;
- эволюцию системы в условиях максимальной независимости от изменений системной платформы;
- доступ к информации любого модуля, в том числе к диагностической информации самой мехатронной системы;
- подключение к внешней сетевой коммуникационной среде;
- использование в архитектуре системы принципов системной интеграции.

Остановимся более подробно на использовании принципов системной интеграции.

Известны принципы реализации тотального информационного сервиса на уровне предприятия, когда интегрируют многочисленные приложения и коммерческие инструментальные средства (базы данных, CAD-CAM системы и др.), чтобы собрать целостную систему. При правильной организации системной интеграции внимание концентрируют на доступе к данным, но не на структурах и типах этих данных.

Таким образом, возникает проблема доступа приложений к данным любого компонента производственной системы. Трудности состоят в бесконечном множестве коммерческих и пользовательских приложений, располагающих собственными интерфейсами и написанных на различных языках программирования. Трудности могут быть преодолены на основе кон-

цепции OLE/COM компании Microsoft. Эта концепция была использована при разработке европейского проекта OPC. Цель проекта состояла в определении стандартной клиент-серверной архитектуры и спецификаций COM-интерфейсов, обеспечивающих унифицированный доступ к данным, независимо от их типа и структуры [17–19]. Таким образом, акцент был сделан на интеграцию, построенную на передаче данных (в том числе управляющих состояниями), а не на прямом управлении компонентами системы.

Обратимся теперь к области ЧПУ мехатронными системами. Основная задача при разработке систем типа PCNC нового поколения состоит в наиболее полном использовании принципов открытой архитектуры. Международные программы OSACA и другие не справились до конца с этой проблемой. Между тем ее решение состоит в использовании лучших достижений системной интеграции больших систем. В самом деле, математическое обеспечение системы ЧПУ содержит оригинальные программные компоненты производителя, компоненты, заказанные у других компаний, готовые коммерческие продукты, компоненты заказчика и конечного пользователя. При этом система должна сохранять все признаки открытой архитектуры. В архитектуре PCNC с неменьшим успехом могут быть использованы принципы OLE/COM и некоторые спецификации OPC, как при разработке отдельных модулей, так и на уровне макропроектирования всей системы в целом.

2.1.2. Модульная архитектура систем ЧПУ на прикладном уровне

Архитектура на прикладном уровне определяется количеством и составом прикладных разделов, называемых задачами управления [20]. В числе подобных задач можно упомянуть:

- геометрическую, ориентированную на управление следящими приводами;
- логическую, организующую управление электроавтоматикой;
- технологическую, гарантирующую поддержание или оптимизацию параметров технологического процесса;
- диспетчеризации, обеспечивающую управление другими задачами на прикладном уровне;
- терминальную, поддерживающую диалог с оператором, отображение состояний системы, редактирование и верификацию управляющих программ.

Структура системы ЧПУ (рис. 31) представляет собой совокупность базовых модулей (обведены сплошными линиями) и дополнительных модулей (обведены пунктирными линиями) [21]. Модули закреплены за задачами управления. К дополнительным модулям отнесены коммерческие приложения. Модуль автономен и является вложенным объектом: он рас-

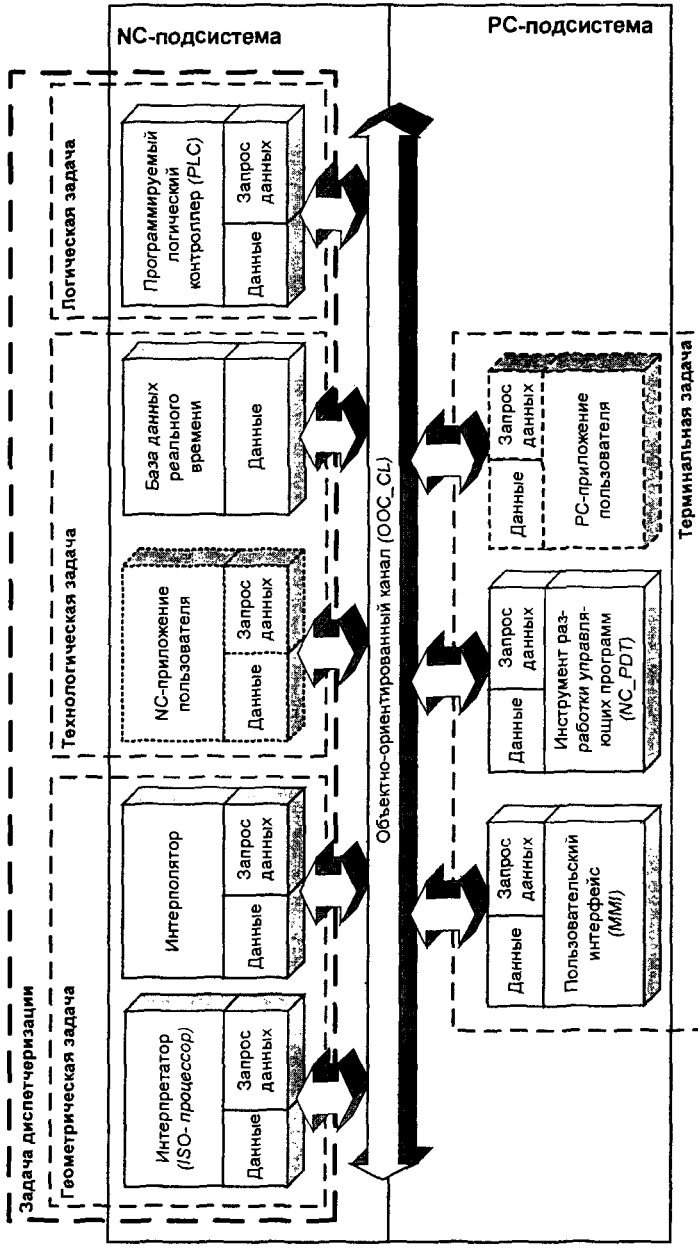


Рис. 31. Модульная архитектура системы ЧПУ типа PCNC и задачи управления

полагает собственными алгоритмической структурой, структурой данных и интерфейсной оболочкой для работы в клиент-серверной среде. Общая структура представлена NC-подсистемой (Numerical Control) и PC-подсистемой (Personal Computer). Первая формирует среду для ЧПУ ориентированных модулей, работающих в реальном времени, и (возможно) для специальных приложений пользователя. Вторая подсистема образует среду Windows-образного интерфейса пользователя и включает инструментальную систему подготовки и тестирования управляющих программ, а также (возможно) другие специальные приложения.

Взаимодействие модулей осуществляется посредством программной объектно-ориентированной магистрали, которая не только поддерживает коммуникационные протоколы, но и выполняет серверные функции. Это значит, что магистраль является глобальным механизмом предоставления модулям информационных услуг. Такая возможность отражена и в самих интерфейсах модулей: они могут предоставлять данные, запрашивать данные, управлять состояниями других модулей. Запрос данных осуществляется синхронным, асинхронным способами или по событию. Выбор механизма запроса зависит от конкретной задачи. При синхронном запросе клиент (модуль, осуществляющий запрос) останавливается в точке запроса и ждет до истечения тайм-аута ответа от сервера (модуля, обслуживающего запрос). При асинхронном запросе клиент продолжает свою работу, а обработка ответа, независимо от времени его получения, выполняется специальной функцией (callback-функцией); ее работа напоминает механизм обработки прерывания. Запрос по событию (синхронный, асинхронный) означает, что ответ будет получен только после изменения данных.

Структура, представленная на рис. 31, обозначает набор модулей, позволяет специфицировать их интерфейсы, выявляет типы запросов, помогает составить техническое задание на объектно-ориентированную магистраль.

2.1.3. Открытая архитектура систем управления

Гибкие и наиболее сложные системы ЧПУ с открытой архитектурой выполняют согласно двухкомпьютерной архитектурной модели (рис. 32, *а*). По мере роста вычислительной мощности компьютеров все более привлекательным становится однокомпьютерный вариант (рис. 32, *б*).

Двухкомпьютерная модель предполагает размещение PC-подсистемы на одном компьютере, а NC-подсистемы – на другом. В PC-подсистеме наиболее целесообразна операционная система Windows NT, а в NC-подсистеме – операционная система реального времени UNIX. Обе операционные системы совместимы в том смысле, что поддерживают коммуникационные протоколы TCP/IP. Это позволяет построить коммуникационную среду, объединяющую подсистемы. Включение в эту среду прикладного

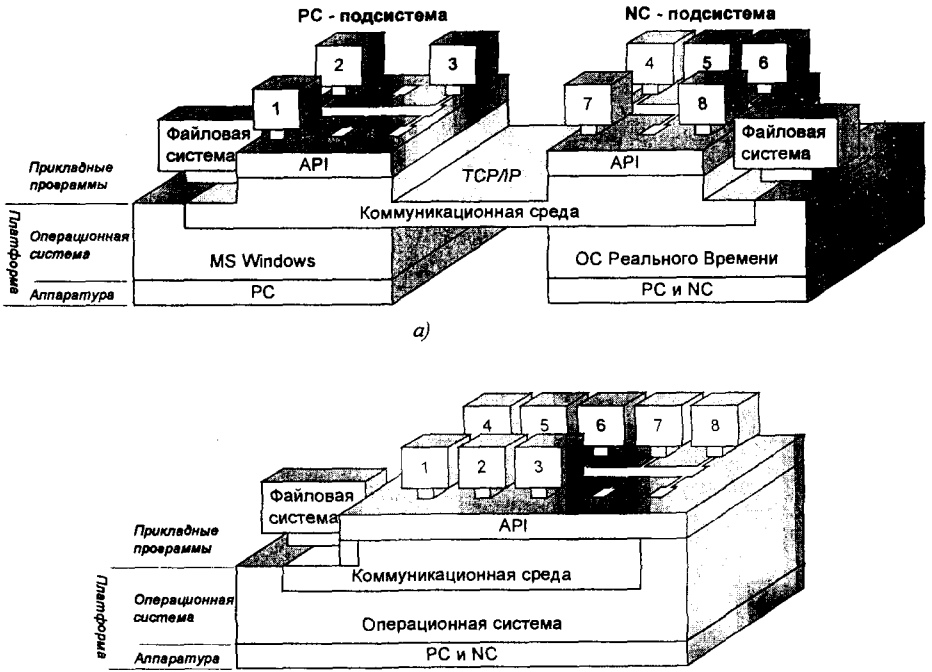


Рис. 32. Открытая архитектура систем типа PCNC: 1 – MMI, 2 – NC_PDT, 3 – PC-приложения, 4 – Интерпретатор, 5 – Интерполятор, 6 – PLC, 7 –База данных, 7 – NC-приложения

уровня с функциями доступа к интерфейсам модулей (а общее число таких функций может достигать нескольких сот) создает виртуальную шину, оказывающую низкоуровневые услуги доступа. Объектная надстройка в шине формирует глобальный сервер, т.е. единую для обеих подсистем объектно-ориентированную магистраль.

Однокомпьютерная модель предполагает использование традиционного компьютера, оснащенного дополнительными контроллерами для связи с мехатронными объектами управления. В их числе могут быть контроллер следящих приводов, программируемый контроллер PLC (Programmable Logic Controller), специальные устройства для управления технологическими процессами и др. В качестве операционной может быть использована система Windows NT, которая, однако, не является системой реального времени и в этой связи требует соответствующего расширения, например в виде системы RTX 4.1 американской фирмы VentureCom.

Система RTX (Real Time eXtention) модифицирует слой HAL (Hardware Abstraction Layer) операционной системы Windows NT и дополняет его диспетчером потоков (threads) реального времени. Диспетчер изолирует прерывания, позволяя строить приложения реального времени, о существовании которых любые другие приложения не подозревают.

Подсистема реального времени RTSS (Real-Time Sub-System) выполняет собственные функции и осуществляет управление ресурсами RTX. Подсистема RTSS реализована в виде драйвера Windows NT, служит дополнением к операционной системе и использует сервисы Windows NT и HAL для работы подсистемы реального времени отдельно от любых других приложений. При этом обычные приложения «видят» подсистему реального времени как устройство (устройства).

Другими компонентами системного уровня являются ядро и драйверы Windows NT. На интерфейсном уровне прикладные программные интерфейсы Win32 и RTX похожи; в них реализованы функции, необходимые соответственно для создания обычных приложений и приложений реального времени.

Разработанную с использованием RTX программу можно отлаживать и запускать также в среде Win32. Однако в RTX есть функции, не имеющие аналогов в Win32, например функции работы с прерываниями.

Архитектурные варианты, показанные на рис. 32, дают общее представление о принципах открытой архитектуры применительно к ЧПУ: четкое разграничение между системным, прикладным и коммуникационным компонентами; возможность независимого развития любого из этих компонентов как на основе оригинальных разработок, так и путем встраивания покупных программных систем; клиент-серверная организация взаимодействия подсистем; стандартизация интерфейсов и транзакций.

2.1.4. Виртуальная модель PC-подсистемы ЧПУ

В вертикальном сечении PC-подсистема имеет многоуровневую структуру (рис. 33) и в полной мере соответствует модели виртуальной машины [22].

Нижний уровень составляет компьютерная аппаратура, выше размещается операционная система Windows NT вместе с драйверами виртуальных устройств (VxD), обеспечивающими управление внешними устройствами, например контроллером панели оператора. Доступ к операционной системе и ее службам осуществляется посредством API-слоя (прикладной интерфейс), который поддержан Win32-функциями и NC-функциями, обеспечивающими вход в подсистемы Windows NT и NC. Функции реализованы в виде DLL (Dynamic Link Library, библиотека с динамическим связыванием). Поверх API-слоя расположен объектно-ориентированный сервер, служащий фундаментом для приложений в системе PCNC.

В числе классов объектов – стандартные из библиотеки MFC (Microsoft Foundation Classes), а также специально разработанные классы OOC_CL объектно-ориентированной магистрали OOC (Object Oriented Channel). Сервер содержит в том числе и общие для всех приложений алгоритмы, такие как обработчики ошибок, средства форматирования и конвертирования данных, управляющие элементы многооконного экрана и др. На прикладном уровне размещаются разнообразные приложения: интерфейс пользователя MMI (Man Machine Interface), инструмент разработки и верификации управляющих программ NC_PDT (NC Program Data Tool) и др.

Заключение

Основными признаками систем ЧПУ нового поколения для мехатронных систем являются принадлежность к классу персональных систем уп-

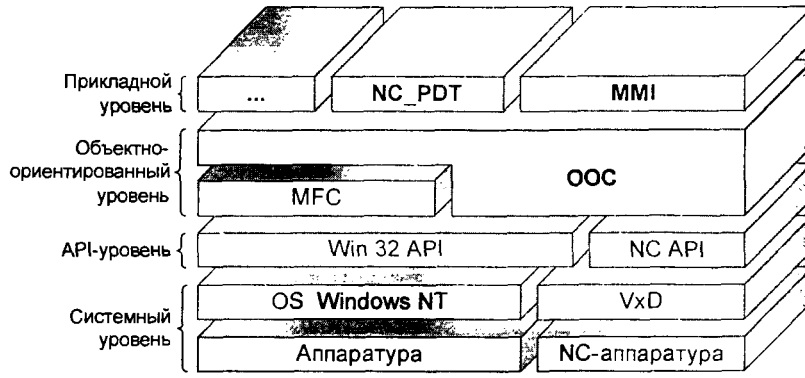


Рис. 33. Виртуальная модель PC-подсистемы

равления PCNC и использование принципов открытой архитектуры. Открытая архитектура предполагает исключительную гибкость (конфигурируемость) системы, использование клиент-серверного подхода в организации транзакций, привлечение объектно-ориентированного подхода к определению макроструктуры, а также на уровне технологии программирования. Все это предопределяет принципиально новую организацию системы ЧПУ, в которой даже модули с традиционными наименованиями имеют новые функциональное и алгоритмическое наполнения, а также и новую программную реализацию. Особо важную роль приобретает PC-подсистема, которая определяет пользовательские характеристики и уровень сервиса для оператора.

2.2. Проблема реального времени в системах управления

Показано, что системы ЧПУ располагают модулями, работающими в машинном масштабе времени, и модулями, работающими в реальном времени. Все прикладные модули взаимодействуют между собой и нуждаются в диспетчеризации, причем проблемы диспетчеризации близки к тем, которые решаются средствами операционных систем реального времени. Раскрыты способы организации совместной работы всех модулей системы управления в целом. Рассмотрены возможности применения стандартных и оригинальных операционных систем реального времени и обосновано решение использования в системе управления расширения реального времени операционной системы Windows NT.

2.2.1. Постановка задачи

Оптимальное использование вычислительных ресурсов систем управления предполагает распределение работы моделей в машинном и реальном масштабах времени. Управление взаимодействием моделей называют диспетчеризацией; она использует средства операционных систем реального времени. Однако диспетчер ни в коей мере не заменяет операционную систему.

Таким образом, поставлена задача найти способы организации совместной работы всех модулей системы управления в целом. Для этого необходимо рациональное решение проблемы реального времени и построение на базе этого решения диспетчера прикладных программ. В этой связи были исследованы существующие предложения по использованию стандартных и оригинальных операционных систем реального времени, а также расширений реального времени операционной системы Windows NT; кроме того, были выделены типы процессов и потоки системы управления.

2.2.2. Реальное время в системе управления

Традиционно системы реального времени, включая модуль диспетчера, строят на базе операционных систем реального времени (ОСРВ). Операционные системы общего назначения, например многопользовательские типа UNIX, ориентированы на оптимальное использование распределения ресурсов компьютера между пользователями и выполняемыми процессами. В системах управления подобные задачи уходят на второй план, поскольку основная цель состоит в своевременной реакции на события в объекте управления. В этой связи рассмотрим классификацию возможных решений.

Исполнительные системы реального времени предлагают разные платформы для разработки и исполнения программного обеспечения. Прикладную часть реального времени разрабатывают на хост-компьютере, затем объединяют с ядром и загружают в систему управления как одну задачу. Такое решение дает высокую точность и быстродействие. Примером может послужить хорошо известная операционная система реального времени VxWorks.

Монолитные ядра реального времени имеют полный набор специфических механизмов реального времени. Ядра компактны, масштабируемы и имеют модульное и хорошо структурированное построение. Типичными представителями служат OS9 (Microwave Systems) и QNX (QNX Software Systems, Канада).

Системы управления с операционной системой UNIX реального времени переписывают ядро стандартной операционной системы с учетом требований реального времени. Такие системы поддерживают весь набор UNIX-приложений. Однако система UNIX реального времени имеет большой объем и низкую реактивность. Типичным и широко используемым представителем семейства UNIX служит операционная система Lynux OS.

Современные системы числового программного управления все чаще используют операционную систему Windows NT с расширением реального времени. Поскольку этот вариант представляется нам чрезвычайно перспективным, мы позднее остановимся на нем подробнее.

2.2.3. Базовые понятия операционной системы реального времени

Система ОСРВ предсказуема в том смысле, что время, затрачиваемое на определенную работу, не должно превышать заранее установленного ограничения. Время реакции на прерывание (interrupt latency) состоит в способности своевременной реакции на внешние события (обычно не превышает 2 – 8 мкс). Время переключения контекста используется для передачи управления от процесса к процессу, от потока к потоку (находится в

пределах 80 – 160 мкс). Время реакции планировщика (scheduling latency) представляет собой задержку активизации процесса после отработки прерывания (находится в пределах 4 – 16 мкс) [23].

В своей работе операционные системы используют набор традиционных механизмов. Механизм приоритетов и диспетчеризации обеспечивает планирование задач реального времени на основе использования некоторого кванта времени (time slice). Механизм межзадачного взаимодействия синхронизирует процессы и передачу данных между ними с использованием семафоров, мютексов, сигналов, событий, разделяемой памяти. Механизм работы с таймерами генерирует прерывания по истечении некоторого настраиваемого интервала времени.

2.2.4. Использование в системах управления операционной системы Windows NT

Windows NT не является операционной системой реального времени, поскольку не имеет достаточного диапазона приоритетов потоков (threads), не позволяет управлять наследованием приоритетов (блокирующий поток должен наследовать приоритет потока, который он блокирует), механизм синхронизации потоков непредсказуем, время реакции на прерывание непредсказуемо.

Между тем в силу растущей популярности в системах управления операционной системы Windows NT проблема как-то должна быть решена. Из всех существующих предложений по реализации ОСРВ на базе Windows NT практическое значение имеют всего два подхода [24].

Первый подход состоит в запуске Windows NT в виде низкоприоритетной задачи операционной системы реального времени (супервизора). При этом предполагается применение ядра классической ОСРВ типа QNX или VxWorks. Существуют решения, в которых в качестве супервизора используется VxWorks.

Второй подход заключается в расширении (в смысле реального времени) Windows NT. Это может быть оригинальная разработка изготовителя системы управления, например система WinCAT (Backhoff Industrie Electronic, ФРГ). Другой вариант – использование готового коммерческого решения, например RTX 4.1 фирмы VenturCom.

Оба подхода имеют свои достоинства и недостатки. Однако подход на базе расширения реального времени для Windows NT все же более перспективен. Во-первых, в расширении использованы те же типы объектов для управления задачами, что и у ядра Windows NT (мютексы, семафоры и т.д.). В противоположность этому VxWorks использует оригинальные функции и механизмы, формирующие собственный стиль, отличный от стиля Windows. Во-вторых, нет необходимости во второй операционной систе-

ме, что сокращает расходы и снимает проблемы установки и стыковки обеих операционных систем на одном персональном компьютере.

Решение в пользу расширения реального времени позволяет быстро обновлять систему управления с появлением новых версий Windows NT, осуществлять мощную защиту приложений, которую Windows выполняет с помощью независимого абстрактного уровня HAL, легко отлаживать коды и использовать возможности стандартных механизмов Microsoft для информационного обмена между Windows и задачами реального времени (IPC – механизм межпроцессной связи, OLE – механизм связывания и внедрения объектов, COM – механизм компонентных моделей, RPC – механизм удаленного вызова процедур).

2.2.5. Стратегия диспетчеризации на базе расширения RTX (Real Time eXtension)

Один из самых надежных и распространенных алгоритмов диспетчеризации в многозадачных операционных системах (ОС) – это алгоритм циклической диспетчеризации, когда для выполнения конкретной задачи предоставляется некоторый квант времени (time slice). По истечении каждого кванта времени планировщик просматривает очередь активных задач и принимает решение, которой из них передать управление [25]. Подобный алгоритм может быть использован в системе числового программного управления. Прежде чем сформировать этот алгоритм, рассмотрим особенности режима времени, в котором работает система управления с операционной системой Windows NT и расширением реального времени.

Понятие о мягком и жестком реальном времени в системе управления

На рис. 34 представлены многоуровневая структура Windows NT с RTX и размещение основных потоков системы управления. Внизу находится уровень аппаратной абстракции реального времени (HAL), где реализованы быстродействующие часы и таймеры, механизм разграничения прерываний между RTX и Windows NT. Подсистема реального времени RTSS выполнена в виде драйвера, работает на уровне ядра Windows NT и обеспечивает основные функции и управление ресурсами RTX. Эта подсистема использует сервисные возможности HAL реального времени и Windows NT для работы с быстрыми часами и таймерами и для обслуживания механизма прерывания. Встроенный в RTSS менеджер потоков (thread manager) и планировщик, основанные на фиксированной системе приоритетов, управляют прикладными задачами [26, 27].

Подсистема RTSS обеспечивает интерфейс между процессами RTX и Windows NT в реальном времени с помощью специального сервисного механизма IPC (Inter Process Communication).

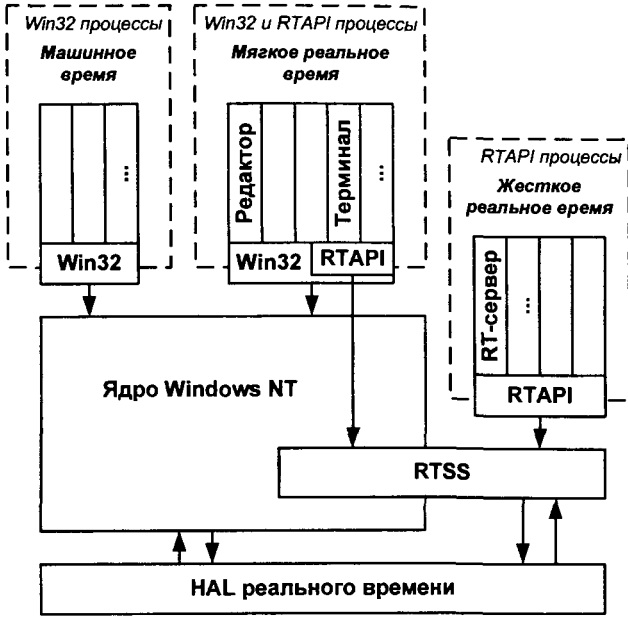


Рис. 34. Основные потоки системы управления с использованием Windows NT и RT

Следует отметить, что RTX полностью соответствует той концепции фирмы Microsoft, которая допускает изменение уровня HAL без изменения ядра операционной системы. При этом расширение функциональностей уровня аппаратной абстракции осуществляется путем добавления драйверов.

В системе работают обычный прикладной интерфейса Win32 для Windows NT (предусмотренный для машинного времени), а также дополнительные прикладные интерфейсы реального времени RTAPI (Real Time Application Interface) и Win32 RT (Real Time). Дополнительные прикладные интерфейсы обеспечивают два режима реального времени: «жесткий» и «мягкий». Это позволяет оптимизировать вычислительные ресурсы системы управления, разделив ее функциональные задачи на три группы:

- в режиме жесткого реального времени решаются критические задачи (интерполяция кадров управляющей программы, ввод-вывод и т.д.), реализованные в процессе RT-сервер (рис. 34);
- в режиме мягкого реального времени решаются задачи, непосредственно связанные с задачами реального времени (например, интерпретация кадра управляющей программы); они реализованы в процессе «терминал» (рис. 34). В отличие от жесткого времени здесь допустимы задержки

Рис. 35. Стратегия диспетчеризации задач реального времени в системе управления

потока из-за свопинга (подкачка), обращения к жесткому диску, прерывания и т.д.;

- в режиме машинного времени работают остальные стандартные прикладные модули системы управления (редактор управляющих программ, встроенная CAM-система, система моделирования процесса обработки и т.д.).

Стратегия диспетчеризации задач в системе управления отображена на рис. 35, где стрелками указана последовательность событий. Первоначально в Windows NT с подсистемой реального времени (RTSS) создается таймер (timer). По истечении кванта времени стандартный механизм генерирует прерывание, которое обрабатывается прикладной call-back функцией или так называемой функцией обратного вызова. Функция реализует алгоритм планирования (диспетчеризации) задач интерпретаций, интерполяции, ввода-вывода, коммуникации и интерфейса оператора MMI. В соответствии с обозначенными для системы управления режимами в жестком реальном времени выполняются задачи диспетчеризации, интерполяции, ввода-вывода, коммуникации. В мягком реальном времени выполняются задачи интерпретации и обновления экранов интерфейса с оператором, а в фоновом процессе – задачи интерфейса с оператором.

2.2.6. Принцип разбиения потоков (threads) в системе управления и схема их диспетчеризации

В системах управления целесообразно придерживаться традиционной для систем реального времени схемы «один процесс – много потоков

(threads)». Схема имеет такие важные достоинства, как быстродействие и высокая реактивность. Высокая реактивность потоков объясняется меньшим временем переключения их контекстов по сравнению с процессами. Потоки используют общее адресное пространство процесса, а процессы нуждаются в разделяемой памяти.

Согласно принятым представлениям о режимах времени в системе управления, выделим три группы потоков:

- жесткого реального времени, работающие в процессе (или в процессах, в зависимости от архитектурного решения) реального времени – это так называемые RTSS-процессы;
- мягкого реального времени, функционирующие в Win32- и RTAPI-процессах;
- машинного времени, работающие в стандартных Win32-процессах.

Обмен данными и синхронизация процессов машинного времени и мягкого реального времени традиционны, это осуществляется на базе общей платформы Win32. Обмен данными между процессами мягкого и жесткого реального времени осуществляется на базе разделяемой памяти (shared memory) – механизма, предоставляемого со стороны RTX.

Процесс RTSS на рис. 36 включает в себя набор потоков, решающих критические задачи в системе управления. Поток диспетчера по сути является call-back функцией таймера (см. рис. 35), в которой реализован планировщик процессов. Планировщик с помощью мютексов или семафоров запускает или останавливает тот или иной поток.

Коммуникационную среду разделим на два потока, один из которых функционирует в режиме жесткого реального времени (поток коммуникационной среды реального времени), а другой работает в режиме мягкого реального времени (поток коммуникационной среды Win32). Задача состоит в передаче данных как между потоками внутри процесса, так и между процессами. Передача данных между потоками коммуникационной среды реального времени и потоками коммуникационной среды Win32, как уже отмечалось, осуществляется посредством разделяемой памяти.

Задача интерполяции реализуется потоком Look Ahead, осуществляющим опережающий просмотр и коррекцию кадров управляющей программы, потоком грубой интерполяции, вызываемым обычно с частотой 50 Гц, и потоком тонкой интерполяции, вызываемым обычно с частотой 1–2 мс, осуществляющим сплайновую интерполяция между точками, рассчитанными в рамках грубой интерполяции. Частота вызова интерполяторов параметрически настраивается в планировщике (в потоке диспетчера реального времени). Поток программируемого контроллера решает задачу управления электроавтоматикой и задачу ввода-вывода.

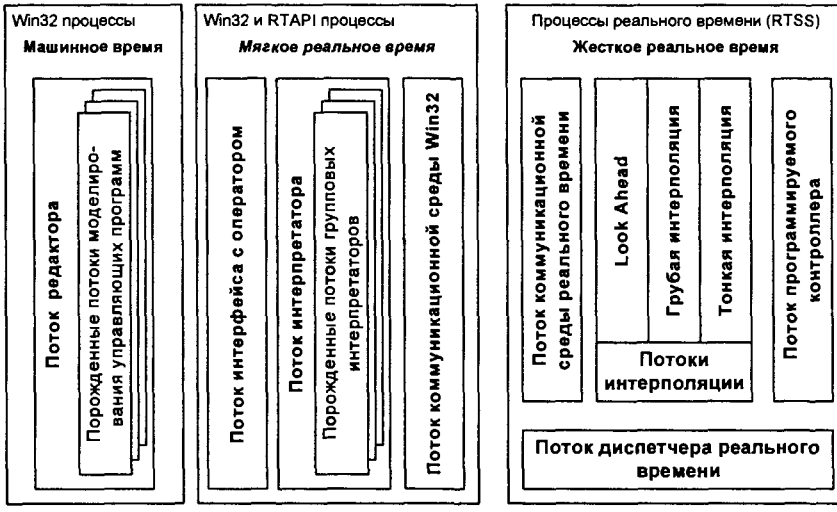


Рис. 36. Потоки системы управления: Look Ahead – просмотр опережающий кадров управляющей программы

В процессе мягкого реального времени, помимо потока коммуникационной среды Win32 и потока интерпретации кадров управляющей программы, работает поток интерфейса с оператором. Поток интерпретатора запускает групповые интерпретаторы как порожденные потоки. В потоке интерфейса с оператором отображаются такие данные процесса реального времени, как текущие координаты, скорость подачи, состояние процесса, режимы системы управления. Отсюда же отправляются управляющие команды процессу реального времени: запуск строки ручного ввода, выбор управляющей программы, «стоп» и т.д.

Процессы машинного времени – это классические Windows-процессы. Примером может послужить редактор управляющих программ, который запускает в качестве порожденных потоки моделирования управляющих программ.

Заключение

Наиболее перспективным подходом при разработке системы управления реального времени в архитектуре Windows NT является использование расширения реального времени. Сложное программное обеспечение системы управления оптимизируется путем разделения выполнения задач в режимах жесткого и мягкого реального времени, а также в фоновом процессе. При этом возникает возможность организации простой и надежной системы диспетчеризации.

2.3. Проблемы управления электроавтоматикой

Рассмотрены варианты управления электроавтоматикой мехатронных систем с помощью программируемых контроллеров и 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 выполняются в одной и той же исполнительной среде; циклы управления инициируются управляющей программой ЧПУ.

2.3.1. Классификация систем управления электроавтоматикой

На рис. 37 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

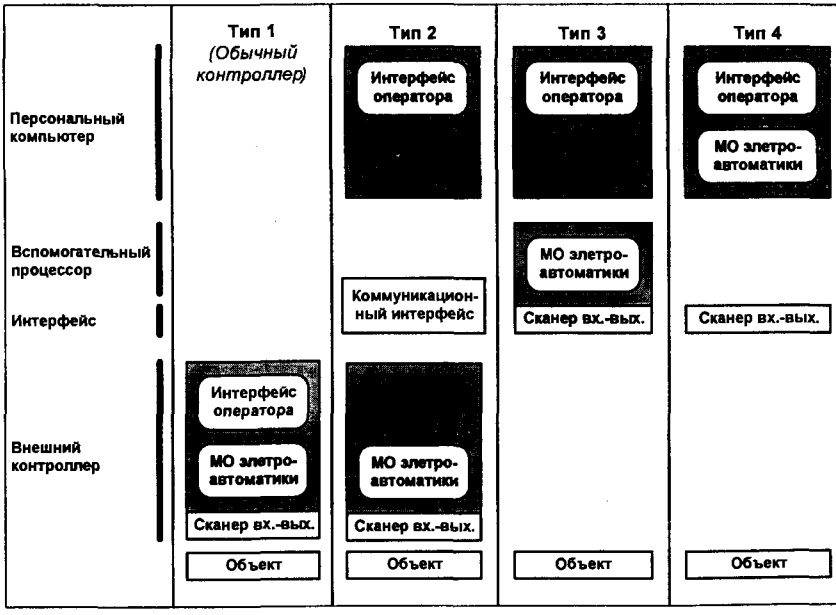


Рис. 37. Классификация программируемых контроллеров

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.

2.3.2. Система понятий, используемых при организации системы управления

Терминология стандарта IEC61131-3 в части организации системы управления ориентирована скорее на традиционные контроллеры. Программная система, удовлетворяющая проекту IEC61131-3, содержит следующие компоненты: конфигурацию, ресурсы, задачи [28–30].

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 выполнения. К таким условиям относятся интервал времени, по истечении которого задача вновь должна быть запущена, или передний фронт события, являющегося глобальной переменной.

Каждой задаче может быть придано несколько запускаемых ею программ. Если задача выполняется в пределах текущего цикла, то и программы будут обработаны в границах этого цикла.

При наличии нескольких задач их совместная работа подчиняется правилам:

- реализуется та задача, для которой справедливы условия выполнения, т.е. закончилось время цикла или соблюдено условие;
- если конкурируют несколько задач, то будет запущена та, которая имеет больший приоритет;

• если конкурируют несколько задач с одинаковым приоритетом, то будет выбрана 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. 38), 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 связыва-

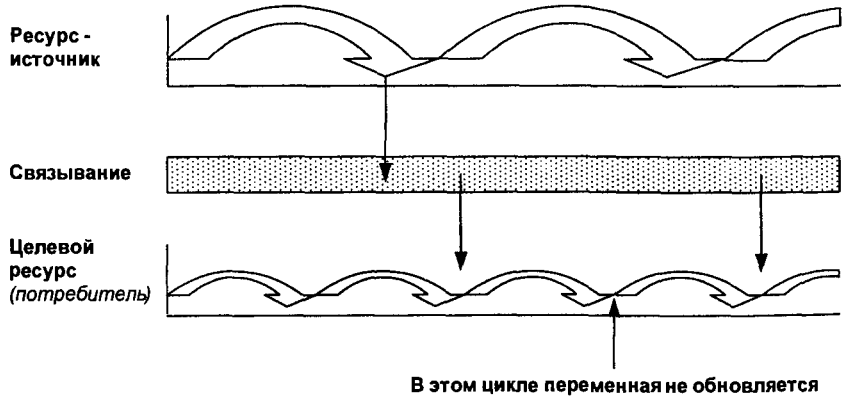


Рис. 38. Межресурсные связи переменных

ющей 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 последний 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.

Временна́я синхронизация двух Virgo посредством механизма связывания показана на рис. 39.

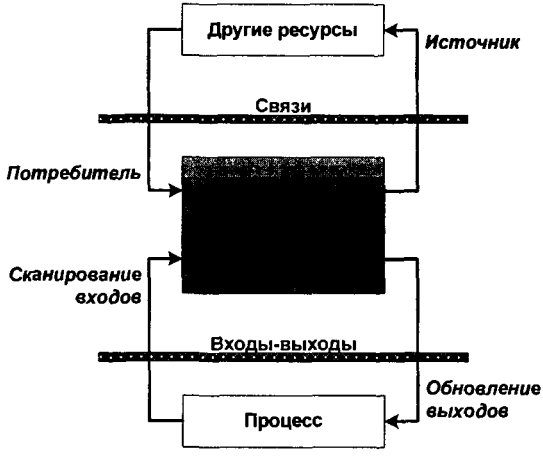


Рис. 39. Временна́я синхронизация двух Virgo

2.3.3. Структура проекта системы управления электроавтоматикой (клиентская часть)

Проект содержит все необходимые компоненты программы управления электроавтоматикой и сохраняется в файле с тем же именем. Он содержит следующие разделы: программные блоки POUs (Program Organization Units), типы данных, элементы визуализации, ресурсы и библиотеки.

Блок POU является структурной единицей программы. В состав POU входят 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. Допустимо, когда одни POUs вызывают другие.

Функция является таким POU, который, будучи обработанным, 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, и при своем объявлении должна получить тип. Это значит, что после имени функции последуют двоеточие и тип. Функции придается результат, т.е. имя функции используется в качестве выходной переменной.

Функция не имеет внутренних условий. Таким образом, при ее вызове с одними и теми же аргументами (входными параметрами), всегда получим один и тот же результат.

Функциональный блок – это такой POU, который выдает во время работы один или более результатов. В отличие от функции он не возвращает значения. Можно создавать репродукции, т. е. экземпляры функциональных блоков. Каждый экземпляр имеет свой собственный идентификатор (имя), а также и 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 доступны извне, но не внутренние переменные.

Декларационная часть функциональных блоков и программ может содержать декларации экземпляров. Доступ к реализации функционального блока в POU, где он был продекларирован, ограничен пока декларация не будет глобальной. Имя экземпляра 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 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. Вызов программы в рамках функции запрещен. Не существует экземпляров программы. Если

POU вызывает программу и в процессе ее работы значения переменных изменяются, эти значения удерживаются к новому вызову программы, даже если в этом новом цикле программа вызвана из другого POU. В этом состоит различие с вызовом функционального блока, для которого изменяются только значения в конкретном экземпляре функционального блока и имеют смысл, когда вызывается тот же экземпляр.

Действия могут быть определены по отношению к функциональным блокам и программам. Они расширяют возможности программирования и могут использовать другой язык. Действие использует данные из тех функциональных блоков и программ, которым оно принадлежит. Действие имеет имя и использует те же входные и выходные переменные и локальные переменные, какие использует 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 свои собственные. Могут быть созданы структуры, перечисленные типы и ссылки.

2.3.4. 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. 40).

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 как управляемая по событию.

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 про-

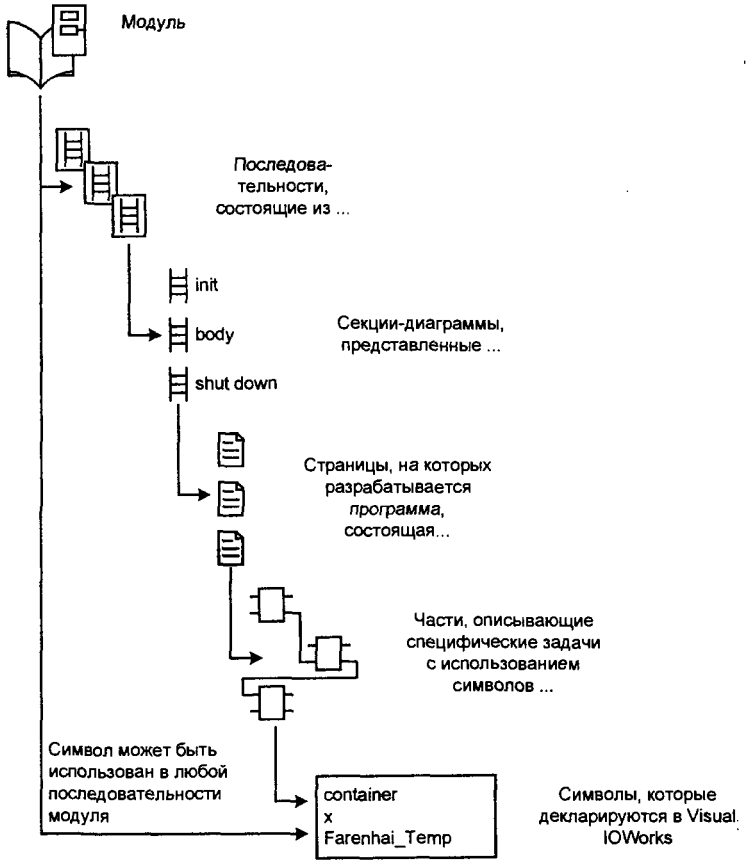


Рис. 40. Взаимоотношение ресурсов в системе Visual IOWorks

грамму. В то же 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: инициализации, тела, закрывающую. 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.

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.

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. 41.

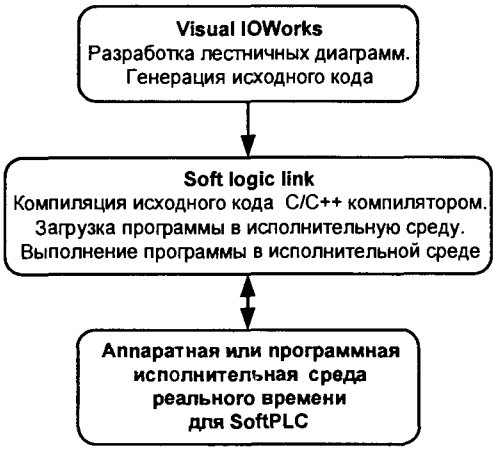


Рис. 41. Укрупненный алгоритм разработки и исполнения программы управления электроавтоматикой (фирма VMIC, США)

2.3.5. Работа серверной части программы управления электроавтоматикой

Алгоритм выполнения программы в нормальном циклическом режиме показан на рис. 42. 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).

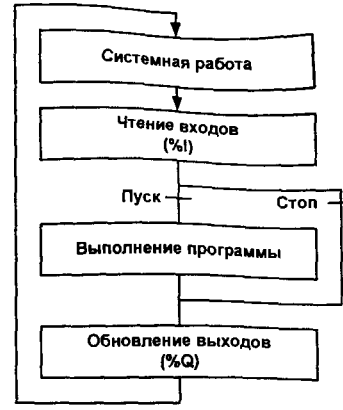


Рис. 42. Циклическая работа программы

Время цикла контролируется сторожевым таймером и не должно превышать определенного значения, например 150 мс, иначе возникает ошибка, останавливающая контроллер. Возможны две ситуации:

1. Время цикла сканирования меньше или равно настройке сторожевого таймера (150 мс). 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.

Диаграмма циклической работы показана на рис. 43.

Другим вариантом выполнения программы служит периодическое управление (рис. 44). В этом случае чтение состояния входов, выполнение программы, обновление выходов осуществляются периодически в соответствии с временем, установленным пользователем при конфигурации (например, от 2 до 150 мс). В начале цикла сканирования контроллера программный таймер устанавливает значение, заданное при конфигурации. Цикл сканирования должен завершиться до истечения данного времени. Если же это время превышено, системный бит %S будет установлен в 1. Время цикла контролируется сторожевым таймером и не должно превы-

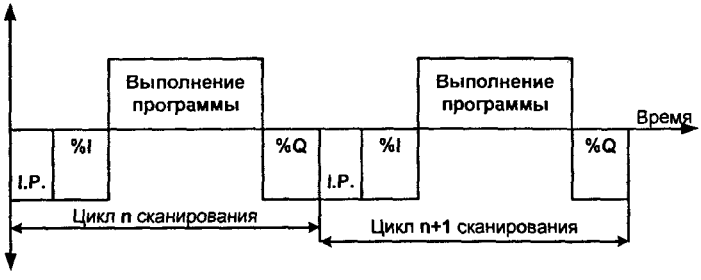


Рис. 43. Диаграмма циклической работы: I.P. – системная работа; %I – чтение входов; %Q – обновление выходов

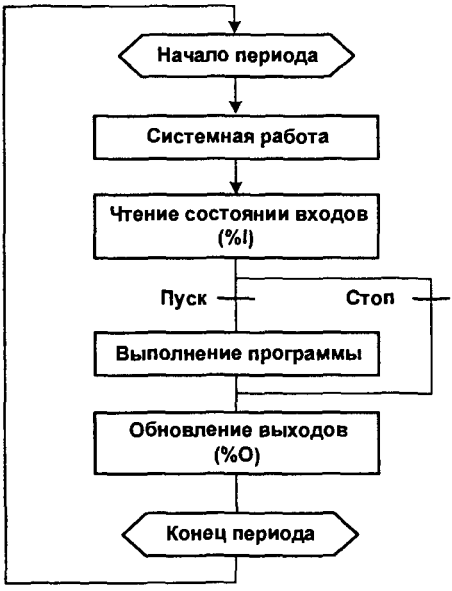


Рис. 44. Периодическое управление

шать 150 мс. Иначе возникает ошибка, останавливающая контроллер. При этом существуют три ситуации:

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

2. Время, установленное при конфигурации, меньше времени сканирования, которое в свою очередь меньше времени сторожевого таймера. Системный бит %S принимает значение 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 становится равным %S = 1.

Диаграмма периодической работы показана на рис. 45.

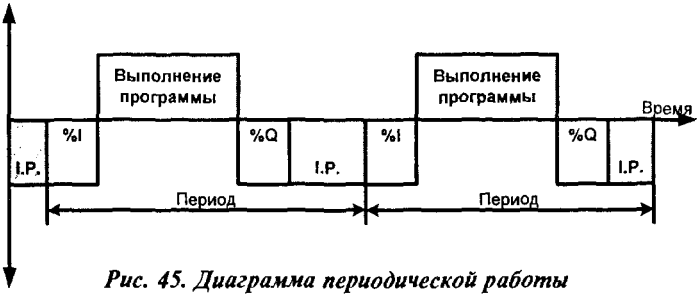


Рис. 45. Диаграмма периодической работы

2.3.6. 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 исполь-

зованию. Для достижения этой цели программисту предлагаются три ключевых решения:

- Объединение функций и данных, которое позволяет разработать 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 использовать 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. 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 при необходимости могут быть перегружены. 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 в разработке классов или 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, PIXELMAP и т.д.

Между клиентскими и серверными данными существует большая разница. Элементы серверных данных уникальны и постоянны в физической памяти. Элементы клиентских данных существуют при визуализации объекта во время сессии редактора. Они дублируются в каждом экземпля-

ре 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. 46.

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 мо-

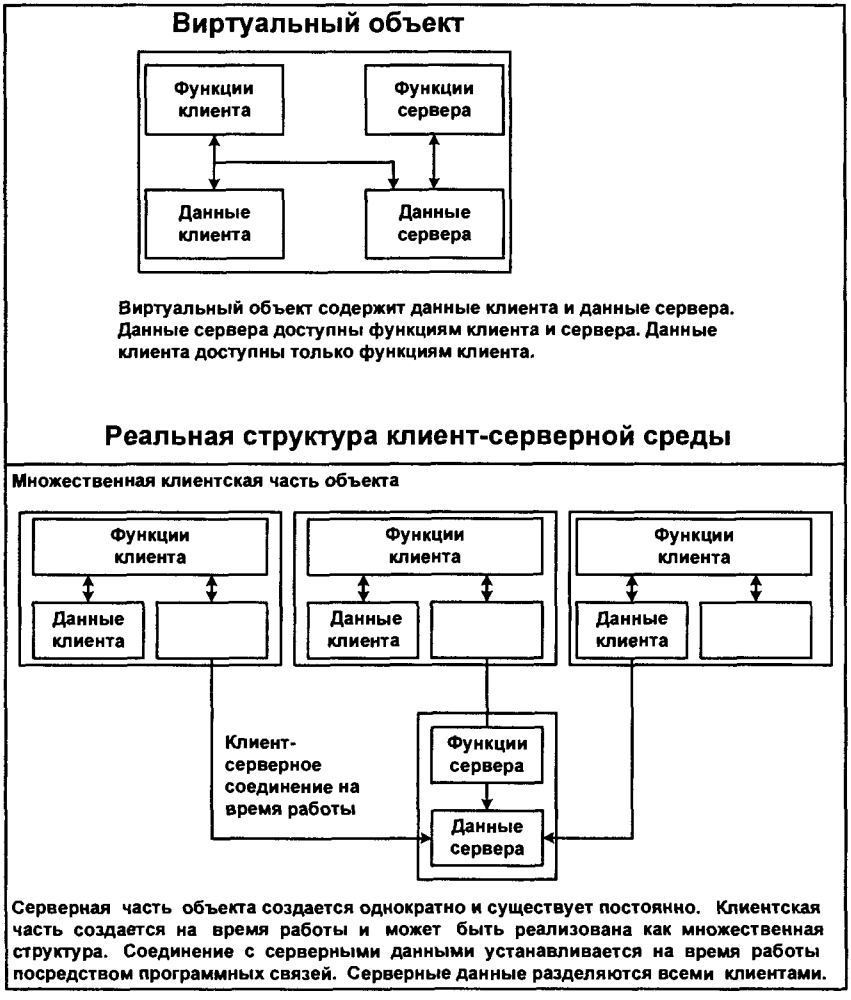


Рис. 46. Клиентские и серверные данные класса

гут быть включены в класс в рамках сессии редактора, процессы системы при этом приобретают соответствующие функции.

Определенный набор библиотечных классов служит для сбора и оценки данных, а также производственной логистики. Для функциональности за пределами стандарта IEC-классов могут добавляться 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. Это имя может быть изменено в любое время. После этого необходимо разработать экземпляр класса, что делается простым его выбором в библиотеке AutomationX. При этом создается автономная копия выбранного класса, или объект. Поведение объекта определяется параметрами, которые задает разработчик класса. После параметризации процесс разработки завершается, верифицируется, готовый файл сохраняется.

Теперь необходимо связать объекты с реальной программой. Необходимо 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.

2.3.7. Особенности управления электроавтоматикой станков с ЧПУ

Первая особенность состоит в том, что в стандарте DIN66025 имена циклов электроавтоматики устанавливают вспомогательные M-функции. Вторая особенность заключается в наличии интерфейса NC <-> PLC между математическим обеспечением ЧПУ и контроллером (программным модулем управления электроавтоматикой). Третья особенность относится к использованию SoftPLC, когда программные обеспечения ЧПУ и контроллера располагаются в единой исполнительной среде.

M-функции дискретны, они включаются и выключаются, активизируются и деактивизируются. При этом передача каких-либо параметров не предусмотрена, эта проблема решается иным образом – с помощью S-, T-, H-функций. Возможны два способа передачи сигнала контроллеру: быстрыми битовыми сигналами или с установлением связи.

Если управляющие M-функции не требуют обратной связи и ожидания выполнения, то можно воспользоваться быстрыми сигнальными битами. Это предполагает присутствие группы битов в интерфейсе NC <-> PLC. Каждая M-функция представлена одним сигнальным битом, который устанавливается и сбрасывается системой ЧПУ. В этом случае система ЧПУ не ждет и не рассматривает никакие сигналы со стороны контроллера. Сиг-

нальные биты остаются в канале интерфейса до тех пор, 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, то 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 (табл. 2). Все М-функции описаны в конфигурационном файле – каждая в своей строке, как показано в табл. 3.

Можно комбинировать правила поведения, чтобы добиться нужной функциональности. Не все комбинации имеют смысл, 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 табл. 4.

Заключение

Система управления электроавтоматикой построена 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 они ориентированы на общепромышленную электроавтоматику и имеют мощную инструментальную поддержку, в которой доминирует объектный подход. В серверной части отдельные задачи реального времени работают циклически, периодически

Таблица 2

Варианты установки битов в управляющей маске для каждой М-функции

Бит	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 3

Описание М-функций в конфигурационном файле

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

Таблица 4

Семантика М-функций

Группа	М-функции	Семантика
Группа завершения программы	М00	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 включаются шпиндель и подача охлаждающей жидкости. Предусмотрена предустановка времени разгона шпинделя.
	М01	Условный останов. Работает как и М00, но при условии подтверждения клавишей на панели оператора.
	М02/М30	Эти инструкции завершают отработку управляющей программы. Шпиндель останавливается, подача охлаждающей жидкости прекращается.
	М99	Конец программы, как и М30, но с дополнительным отводом исполнительных органов в безопасную позицию. Безопасная позиция определяется машинными параметрами
Группа управления шпинделем	М03/М04	Включение вращения шпинделя по/против часовой стрелки. После предустановленного времени разгона начинается отработка других функций кадра. Функция деактивируется либо командой М05 (останов шпинделя), либо командой завершения программы.
	М05	Останов шпинделя, который произойдет, когда все инструкции кадра выполнены.
	М19	С помощью функций М19 S<...> можно остановить шпиндель в ориентированном положении, причем S-функция указывает угол поворота в градусах. Например: М19 S90. Конечно, это возможно, если используется следящий привод управления шпинделем.
Группа функций охлаждения	М07	Включение функции охлаждения с идентификатором 1. Функция работает перед началом выполнения кадра и выключается либо функцией М09 (выключение охлаждения), либо функцией завершения программы.
	М08	Включение функции охлаждения с идентификатором 2.
	М09	Выключение охлаждения после того, как все функции кадра отработаны.
Группа смены инструмента	М06	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 остается выключенным.

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 выполняются в одной и той же исполнительной среде. Еще одна особенность состоит в том, что циклы управления электроавтоматикой вызываются из управляющей программы. Программирование этих циклов, по-видимому, требует своей объектно-ориентированной поддержки.

2.4. Построение межмодульной коммуникационной среды

Рассмотрен принцип построения коммуникационной среды систем ЧПУ, при котором коммуникационная среда берет на себя проблему интеграции всех модулей системы управления и проблему межмодульной коммуникации. Компонентный COM-подход и известные принципы системной интеграции использованы при разработке отдельных модулей системы ЧПУ и на уровне ее макропроектирования, т.е. проблема межмодульной коммуникации решается так же, как и проблема системной интеграции. Компонентный COM-подход поддерживает распределенную систему функционирования, когда модули системы ЧПУ могут работать в разных потоках (threads) и разных системах, а также выступать в качестве COM-серверов (компонентов) и COM-клиентов.

Традиционно коммуникационную среду системы ЧПУ трактуют как некоторый набор интерфейсных API-функций (Application Program Interface functions, функции прикладного интерфейса) для обмена данными с ядром системы ЧПУ, при этом общее число API-функций может достигать нескольких сот. При таком подходе, однако, любое изменение в архитектуре системы требует немалых усилий разработчиков. Таким образом, существующая ситуация состоит в том, что API задает некоторый общий интерфейс подключения модулей в системе ЧПУ, но не поддерживает их интеграцию. Решение проблемы следует искать в использовании продвинутых технологий фирмы Microsoft.

Посмотрим, как выглядит интерфейс Win32 API, обеспечивающий доступ к операционной системе Windows NT (рис. 47, *а*).

Непосредственный доступ осуществляется с помощью Win 32 API-функций. Более высокий уровень сервиса для доступа к операционной системе обеспечивается с помощью классов библиотеки MFC, что уменьшает время разработки приложений. На базе объектов выстраивают COM

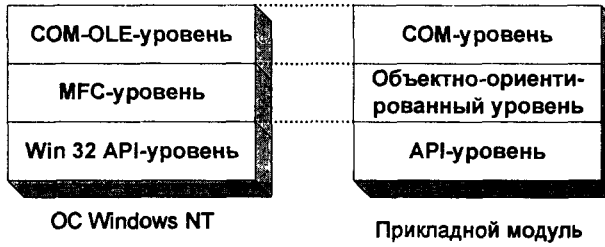


Рис. 47. Модели прикладного интерфейса: а – интерфейс Win 32 API, обеспечивающий доступ к операционной системе Windows NT; б – интерфейс прикладных программ, обеспечивающий доступ к коммуникационной среде

(Component Object Model) и OLE (Object Linking and Embedding) механизмы третьего уровня, которые предполагают соответствующую структуру интерфейсов прикладных программ [31, 32].

Подход, предлагаемый для построения коммуникационной среды системы ЧПУ, заключается в том, что используется аналогичная трехуровневая модель (рис. 47, *б*). При этом коммуникационная среда берет на себя проблему интеграции модулей системы PCNC и проблему межмодульной коммуникации. Компонентный COM-подход и известные принципы системной интеграции могут быть использованы не только при разработке отдельных модулей системы ЧПУ, но и на уровне ее макропроектирования. Последнее означает, что проблема межмодульной коммуникации решается так же, как и проблема системной интеграции. Модули, реализованные в виде компонентов, можно отключать и изменять без перекомпиляции программного обеспечения и без перекомпоновки системы ЧПУ, при этом меняется поведение системы, но не ее архитектура. Компонентный COM-подход поддерживает распределенную систему функционирования: модули системы ЧПУ могут работать в разных потоках (threads) и разных системах и выступать в качестве COM-серверов (компонентов) и COM-клиентов.

2.4.1. Базовые функции коммуникационной среды

Введем понятие объектно-ориентированной магистрали как средства межмодульной коммуникации и источника необходимых межмодульных услуг.

Магистраль является программным (виртуальным) каналом для обмена данными между подключенными к каналу модулями. Объектно-ориентированная магистраль предполагает наличие в ее программном обеспечении набора объектов, решающих задачи подключения модулей, транс-

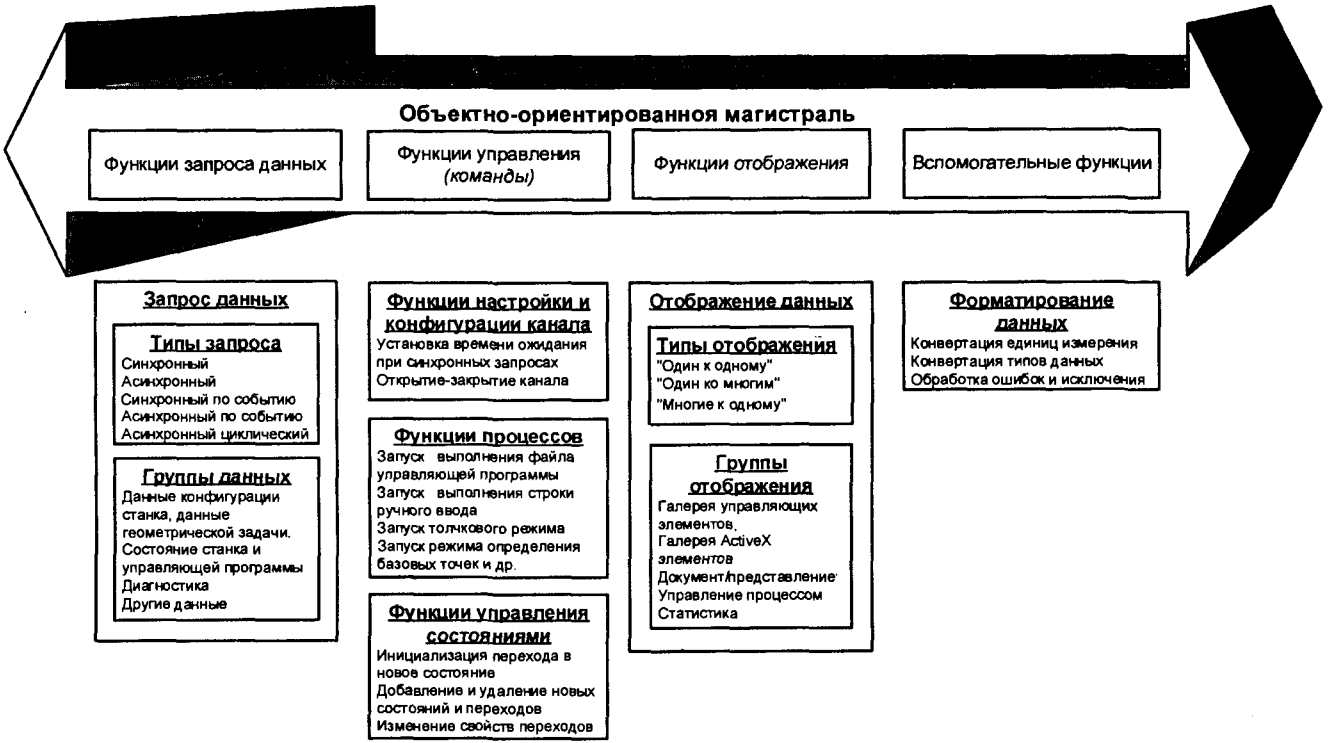


Рис. 48. Функции интерфейса объектно-ориентированной магистрали

портировки данных и др. [33]. На рис. 48 выделены четыре типа функций объектно-ориентированной магистрали: запроса данных, управления, отображения, вспомогательные.

Функции запроса данных предполагают, что в системе PCNC существуют данные разных типов и потребность в них различна. Например, данные о количестве и именах используемых на станке координатных осей требуются один раз в момент инициализации системы. Данные о текущем состоянии выполняемого процесса нужны постоянно, чтобы принимать корректные решения. Существуют и другие варианты запросов на получение данных. Поэтому объектно-ориентированная магистраль предусматривает пять их видов: синхронный, асинхронный, синхронный по событию, асинхронный по событию, асинхронный циклический запрос. Группы запрашиваемых данных примерно совпадают с группами API-функций.

Функции управления можно разбить на три группы:

- управления каналом, открывающие и закрывающие канал;
- процессов, контролирующие ход их выполнения, включая запуск и останов;
- управления состояниями, которые будут рассмотрены ниже.

Если запрос связан с процессом получения данных из модуля-источника (сервера) через внутреннюю структуру коммуникационной среды, то процесс переноса и обработки этих данных в модуль-клиент относится к группе функций отображения данных.

На рис. 49 выделены фазы обмена данными через магистраль. В фазе запроса данных определяется сервер данных. Данными могут быть: текущие координаты, величина подачи, список активных G-функций (G-вектор) и т.д. В этой же фазе устанавливаются тип запроса и приемник данных (клиент).

Запрос данных	Отображение данных
• Источник данных *(сервер данных)* • Сессия запроса • Приемник данных *(клиент)*.	• Тип отображения • Формат отображения.

Рис. 49. Две фазы обмена данными через объектно-ориентированную магистраль

В фазе отображения определяются тип и формат отображения. Формат отображения предполагает, что одни и те же данные могут быть представлены, например, в виде пиктограмм, текста или чисел.

Объектно-ориентированная магистраль предусматривает три типа отображений, каждый из которых поддерживается собственным механизмом. В зависимости от способа вывода данных отображения разделены на несколько групп:

- визуализация в галерее управляющих элементов (control elements, терминология Microsoft), которая строится на базе стандартных Windows-элементов;
- визуализация в галерее ActiveX-элементов (терминология Microsoft), которая строится на базе OLE-элементов Windows, расширяющих стандартный набор Windows-элементов [34];
- визуализация в среде «документа-представления» (терминология Microsoft) при создании пользовательских приложений на базе стандартного механизма MFC;
- визуализация с целью управления ходом процесса в системе PCNC;
- статистическое накопление данных для сохранения, например, в базе данных или их последующего анализа.

Вспомогательные функции составляют единый механизм конвертирования и форматирования данных, обработки ошибок, формирования исключений (exceptions) для всех модулей, подключенных к объектно-ориентированной магистрали.

2.4.2. Клиент-серверные транзакции при запросе данных

Взаимодействие модулей системы PCNC носит клиент-серверный характер [22]. Транзакции (сессии) между модулем-клиентом, запрашивающим услугу, и модулем-сервером, оказывающим услугу, обобщены по их назначению на рис. 50.

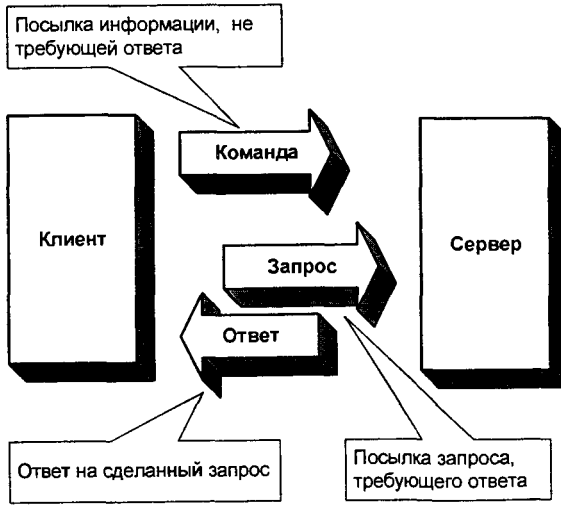


Рис. 50. Клиент-серверные отношения в системе PCNC

Команда направляется серверу для выполнения некоторого действия, например запуска управляющей программы. Такая команда не предполагает ответа со стороны сервера. Другой вариант: запрос направляется серверу с целью получения некоторых данных, например значений текущих координат.

Такой запрос предполагает ответ со стороны сервера. На рис. 51 показаны базовые транзакции: синхронная, асинхронная и по событию. В рамках синхронной сессии (рис. 51, *а*) клиент направляет запрос серверу и приостанавливает работу в точке запроса. При готовности сервер отвечает, после чего клиент продолжает работу. В рамках асинхронной сессии (рис. 51, *б*) клиент направляет запрос серверу и продолжает свою работу. Ответ сервера обрабатывается специальной callback функцией (аналогичной функции обработки прерывания) клиента. Событием в системе PCNC служит всякое изменение данных, например изменение состояния процесса.

В рамках асинхронной сессии по событию (рис. 51, *в*) клиент направляет запрос серверу и продолжает свою работу. Сервер отвечает лишь после того, как произойдет событие, т.е. изменятся запрашиваемые данные. Ответ обрабатывается callback функцией клиента.

Синхронную сессию по событию (рис. 51, *г*) используют только для отладки системы PCNC. В этом случае клиент направляет запрос серверу и приостанавливается в точке запроса. Клиент продолжит свою работу в том случае, если произойдет событие и сервер ответит клиенту.

На основе базовых транзакций может быть реализован циклический опрос данных, например постоянный опрос текущих значений координат для вывода их на экран (рис. 52). Первоначально приходит текущее значение координат в рамках асинхронного запроса, после чего инициируется асинхронный запрос по событию. Когда данные изменятся, сервер ответит и ответ будет обработан callback функцией. Callback функция осуществит очередной асинхронный запрос по событию. Таким образом, опрос данных становится циклическим – клиент будет получать ответ от сервера всякий раз после изменения данных.

Организация транзакций в системе PCNC на основе предложенных сессий позволяет оптимизировать трафик коммуникационной среды и экономично использовать ресурсы для выполнения других задач.

Определение схемы отображения отслеживаемых данных. Во второй фазе обмена данными через коммуникационную среду в фазе отображения (см. выше) предлагается использовать три схемы (рис. 53):

- «один к одному», когда один сервер отображает значения своих данных в одном клиенте;
- «один к многим», когда один сервер отображает значения своих данных в нескольких клиентах;

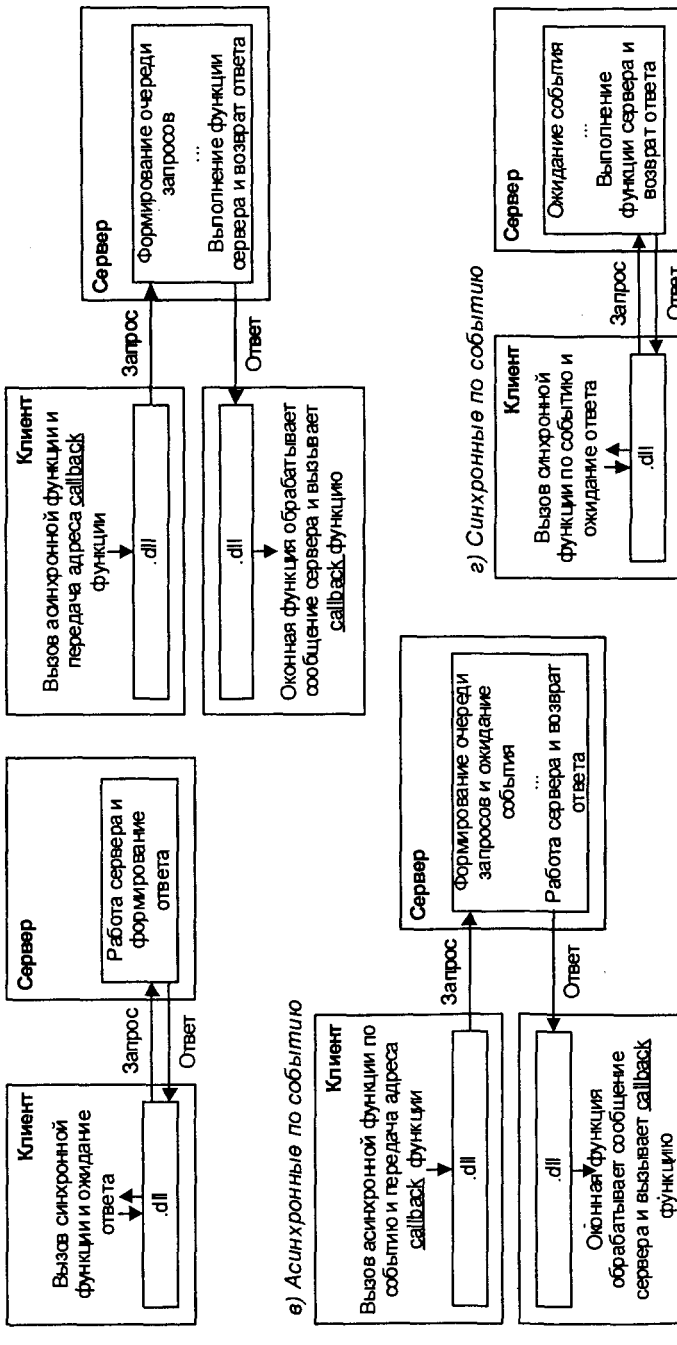


Рис. 51. Основные сессии обмена данными

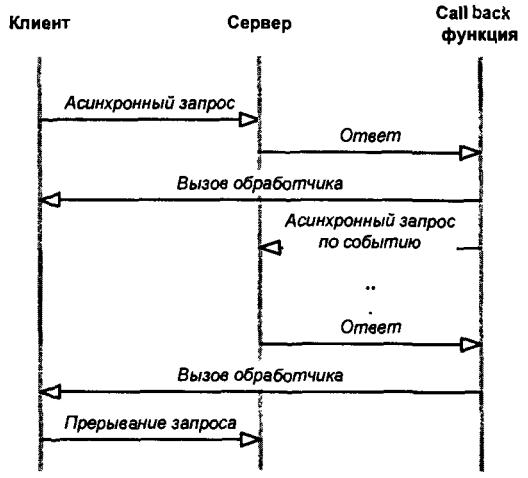


Рис. 52. Диаграмма циклического опроса данных

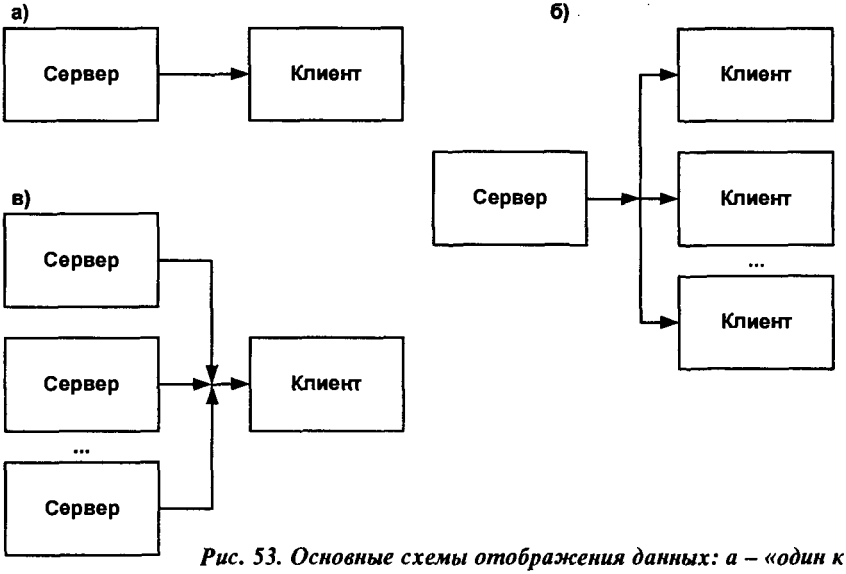


Рис. 53. Основные схемы отображения данных: а – «один к одному»; б – «один к многим»; в – «многие к одному»

• «многие к одному», когда несколько серверов отображают значения своих данных в одном клиенте.

Другие схемы, как, например, «многие к многим», представляют собой комбинацию указанных ранее.

Формализация процедуры обмена данными позволила выбрать основные классы коммуникации в терминах объектов. Сервер представлен классом-источником абстрактных данных CAbstractData. Клиент представлен классом-получателем данных CReceiver. Отношения, в которые вступают эти классы, определяют тип транзакции и схему отображения.

Применение схемы «один к одному» подразумевает, что для одних и тех же данных, например выводимых на экран, будут создаваться разные объекты при переключении экранов. При этом трафик в коммуникационной среде будет определяться данными, визуализируемыми на текущем экране. Это означает, что в схеме «один к одному» объект CAbstractData не может существовать без объекта CReceiver.

Схема «один к многим» предполагает, что объект CAbstractData существует независимо от объекта CReceiver, поскольку определенный тип данных запрашивается одним и тем же объектом CAbstractData, но отображается разными объектами CReceiver. Эту схему целесообразно использовать для отображения тех данных, за которыми ведется постоянное наблюдение, вне зависимости от выбора текущего экрана. Примером служит состояние интерполятора, которое может быть выведено на некоторые из экранов интерфейса оператора.

Схема «многие к одному» используется при отображении данных, вычисляемых на основе значений нескольких объектов CAbstractData. Типичный пример – вычисление процентого отношения скорости подачи, которое требует запрограммированного и текущего значений скорости подачи. Эта схема может быть также применена, когда необходима начальная инициализация и нужно определить, например, количество и имена координатных осей для отображения на экране текущих позиций приводов. Для реализации указанных транзакций и схемы отображения данных была разработана объектно-ориентированная модель коммуникационной магистрали.

2.4.3. Виртуальная структура объектно-ориентированной магистрали

Виртуальная структура показана на рис. 54 в виде трехуровневой модели классов. На уровне запроса (базовом нижнем уровне) размещены коммуникационные классы, ответственные за доставку информации в рамках той или иной транзакции. Классы процессов этого же уровня подготавливают и запускают процессы отдельных режимов системы ЧПУ. Вспомогательные классы базового уровня осуществляют конвертирование и форматирование данных, а также занимаются обработкой ошибок и исключе-

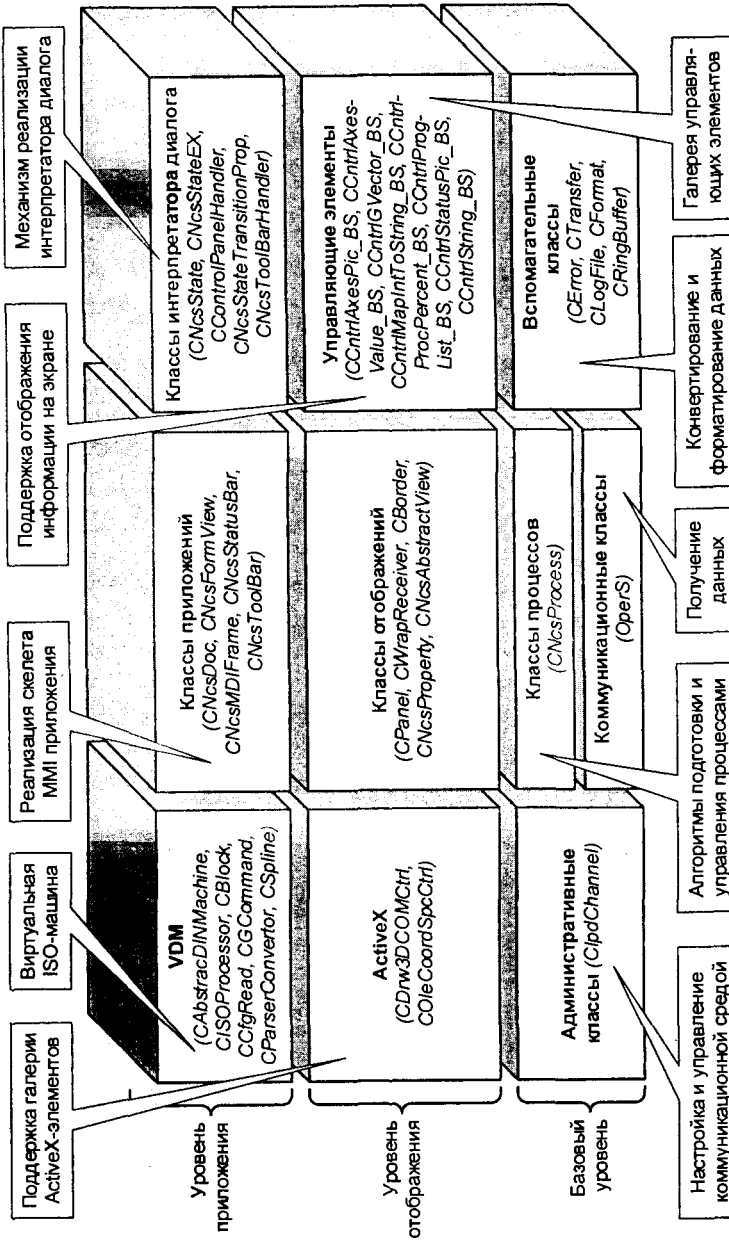


Рис. 54. Виртуальная структура коммуникационной среды в виде объектно-ориентированной магистрали

ний, ведением журнала событий. Административные классы базового уровня управляют коммуникационной средой и осуществляют ее настройку.

Уровень отображения предназначен в основном для вывода информации на экран (для визуализации). Ведущую роль здесь играют классы отображения, устанавливающие связь между управляющими элементами экрана и коммуникационными классами. Классы управляющих элементов экрана служат для приема конкретной информации в определенной форме, а галерея этих классов определяет дизайн экрана и меру богатства его изобразительных средств. Классы ActiveX-управляющих элементов также составляют галерею и относятся к OLE-механизмам Windows NT. Их применение сохраняет возможности обычных управляющих элементов, а кроме того, позволяет использовать различного рода настройки (масштаб, цвета, шрифты и др.) и создавать экзотические элементы управления, например непрямоугольные или прозрачные формы на экране.

Прикладной уровень поставляет приложениям такие работающие с коммуникационной средой классы, как «документ–представление», «машина состояний» и др. Классы интерпретатора диалога с оператором реализуют механизм управления состояниями системы PCNC, воздействуя на «машину состояний». Виртуальная ISO-машина предоставляет набор классов, реализующих механизмы синтаксического и семантического контроля кадра управляющей программы на языке ISO-7bit, а также интерпретацию и конвертирование кадров в IPD-коды интерполяции (InterPolator Data).

Обозначенная подобным образом структура объектно-ориентированной магистрали с иерархическим построением уровней и блоками с четко установленной функциональностью упрощает переход к практическому построению коммуникационной среды.

2.4.4. Организация коммуникационной среды в виде открытой модульной системы

Для построения открытой коммуникационной среды необходимо следующее:

- использование таких услуг, как конвертирование и форматирование данных, обработка ошибок и ведение журнала событий (log-файла);
- адаптация к конкретной системной платформе;
- расширение галереи управляющих элементов и галереи ActiveX-элементов непосредственно у пользователя для решения конкретных задач;
- расширение группы коммуникационных классов для обслуживания специальных модулей (например, модуля управления лазером, модуля отладки управляющих программ, написанных на высокоуровневом языке, и т.д.);

• использование специальных классов прикладного уровня (например, классов документов или шаблонов документов) для реализации пользовательских режимов в интерфейсе оператора.

Конфигурируемая коммуникационная среда решает проблемы настройки на конкретный объект управления (станок). Конфигурируемость обеспечивается путем выделения в коммуникационной среде модулей, реализуемых в виде библиотек DLL.

Пример структуры коммуникационной среды, соответствующей ее виртуальной структуре на рис. 54, представлен на рис. 55. В нижней части рисунка находится независимый уровень (IndNcsCl.dll), который не зависит от системной платформы и других модулей коммуникационной среды. Помимо вспомогательных классов здесь реализованы абстрактные классы CAbstractData и CReceiver, служащие ядром для построения механизмов запроса и отображения данных.

Выше расположен базовый уровень (BasNcsCl.dll), осуществляющий адаптацию к конкретной системной платформе. Здесь же находятся административные классы. За базовым уровнем следует коммуникационный, в составе которого имеются модули управления движением и процессами. Базовый уровень служит основой для построения коммуникационного уровня, где реализованы такие модули, как управление движением (McoNcsCl.dll), управление процессами запроса данных SavNcsCl.dll, диагностика (DgnNcsCl.dll) и т.д.

На основе независимого уровня также выстроены галереи управляющих элементов CtrlGall.dll и ActiveX-элементов, осуществляющих отобра-

Рис. 55. Пример модульной структуры коммуникационной среды

жение данных. Прикладной уровень (AppNcsCl) использует все обозначенные модули для реализации классов приложения и классов управления состояниями.

Представленная коммуникационная среда позволяет компоновать систему PCNC лишь из необходимых модулей, соответствующих конкретным задачам объекта управления.

Заключение

В процедуре обмена данными в системе ЧПУ типа PCNC выделены фазы запроса и отображения данных. Производители систем ЧПУ ответственны за обеспечение открытой структуры в первой фазе, а станкостроители и конечные пользователи располагают возможностями открытой системы для решения своих специальных проблем во второй фазе. Объектно-ориентированная магистраль – это не только коммуникационная среда для обмена данными, но и единый сервер, оказывающий услуги любым подключенным к ней модулям. Организация транзакций по типу синхронной, асинхронной сессии и сессии по событию позволяет минимизировать трафик в коммуникационной среде и оптимально использовать ресурсы процессора. Компонентный подход и принципы системной интеграции могут быть использованы не только при программировании системы, но и на уровне ее макропроектирования.

2.5. Принципы построения удаленных терминалов ЧПУ

Существуют ситуации, когда построение распределенных систем ЧПУ создает дополнительные удобства, а также ситуации, когда без этого обойтись невозможно. В обоих случаях применяют удаленный (от ядра ЧПУ) терминал, который дублирует основной терминал системы ЧПУ или заменяет его. Особенность удаленного терминала состоит в том, что он может использовать иную платформу, его средства визуализации и управления более лаконичны. При этом должен осуществляться доступ к основным функциям ядра ЧПУ через локальную или корпоративную сети, а быть может, через Интернет. Рассмотрены принципы разработки удаленных терминалов, которые являются новыми компонентами распределенных систем управления.

Новые требования к системам управления состоят в том, что они все более приобретают распределенный характер. Существенно меняется роль оператора в зоне управления. Особое значение придается возможности подключения удаленных терминалов для получения необходимой информа-

ции «снизу» и использования Интернета. Соответственно так называемой технологии «тонкого клиента» (thin client) в качестве web-сервера может выступать сама система ЧПУ. Все эти проблемы были затронуты в рамках исследования, связанного с созданием типового активного удаленного терминала с применением языка Java [35].

2.5.1. Удаленный терминал в системе управления

Удаленный терминал предполагает отделение терминальной задачи ЧПУ от всех остальных или ее дублирование. При этом ядро системы управления располагают, исходя из конструктивных соображений, а терминальную часть – на другом компьютере сети, т.е. там, где это удобно с позиций организации управления. Сеть при этом может быть локальной сетью участка или цеха, Интернет-сетью предприятия или глобальной Интернет-сетью. Удаленный компьютер может иметь платформу, отличную от платформы компьютерной системы ЧПУ.

Необходимость удаленного терминала обусловлена следующими соображениями. В процессе работы оператор системы ЧПУ должен следить за информацией о ходе технологического процесса, текущими координатами приводов подачи, сообщениями об ошибках в системе управления и т.д. Между тем современные станки и технологические линии нередко имеют протяженность, превышающую сотню метров. Традиционное решение состоит в том, что оператор непрерывно перемещается в зоне оборудования, обращаясь к специальным пультам, распределенным по длине рабочего участка. Подобный подход требует значительных затрат и увеличивает площадь рабочего пространства оператора. Альтернативой служит применение в качестве удаленных терминалов портативных компьютеров «notebook» или карманных компьютеров типа Palm, которые могут быть переносными. Для подключения удаленного терминала в различных точках рабочей зоны (через 30 – 50 м) имеются разъемы. При этом оператор может выбрать для себя наиболее комфортную позицию. Организация математического обеспечения удаленного терминала требует разработки новой концепции.

2.5.2. Информационные технологии, используемые при создании удаленного терминала

Рассмотрим некоторые информационные технологии, которые наиболее удобно применять для решения поставленной задачи.

Во-первых, нас интересует технология «тонкий клиент/сервер», которая предполагает, что запуск и работа приложения, а также управление приложением происходят на сервере (в нашем случае это устройство ЧПУ) [36]. Эта модель использует многопользовательскую операционную сис-

тему и технологию передачи полного пользовательского интерфейса на удаленное устройство пользователя. Высокоэффективный протокол «представления Windows» отделяет работу приложений от удаленного терминала и посылает по сети только события клавиатуры и мыши, а также обновления изображений на экране.

Во-вторых, нас интересует объектно-ориентированный язык Java компании «Sun Microsystems», который уже изначально располагал высокой степенью переносимости при разработке распределенных сетевых приложений. Переносимость достигается использованием виртуальных машин, интерпретирующих байт-коды на разных аппаратных платформах и в разных операционных системах. Апплеты Java служат разновидностью приложений, интерпретируемых виртуальной Java-машиной, встроенной в среду Java-совместимых браузеров. Поскольку web-браузеры разрабатывались для отображения HTML-документов, работа апплетов Java в среде браузера предполагает использование HTML-тэга <APPLET>, вызывающего апплеты.

Схема функционирования апплета выглядит так. Апплеты сохраняются на Интернет-сервере; они загружаются на разные клиентские платформы и выполняются браузером клиентской машины. Загрузка и исполнение осуществляются под надзором системы безопасности, которая защищает от выполнения недопустимых операций. Если браузер обнаруживает HTML-страницу с апплетом, он запускает Java-машину и передает ей информацию <APPLET> тэга. Загрузчик, находящийся внутри Java, отыскивает необходимые классы для выполнения апплета. Как часть процесса загрузки запускается верификатор, проверяющий корректность класса и надежность его кода.

Апплеты позволяют выполнять сложную обработку данных, полученных от сервера. Из соображений безопасности апплеты не имеют доступа к файловой системе локального компьютера. Данные поступают только от сервера. Для повышения производительности Java-приложений в современных браузерах используют компиляцию «на лету» (Just-In-Time compilation, JIT).

При первой загрузке код апплета транслируется в исполняемую программу, которая сохраняется на диске. Таким образом, JIT-компилятор преобразует байт-коды в команды процессора целевой машины непосредственно перед выполнением. В результате скорость апплета увеличивается в несколько раз [35].

Высказанные соображения были положены в основу разработки удаленного терминала. На рис. 56 представлена принципиальная схема его подключения. Для связи удаленного терминала с системой ЧПУ использовано стандартное Интернет-соединение. В сравнении с традиционным ин-

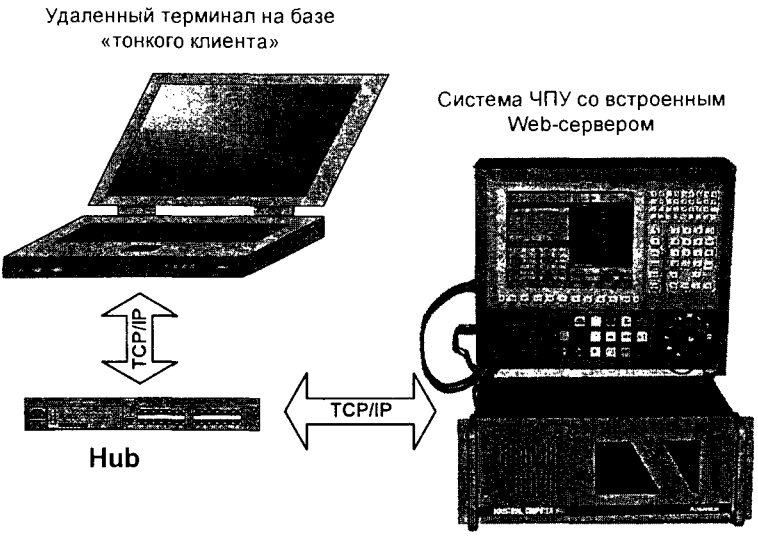


Рис. 56. Схема подключения (функционирования) удаленного терминала

терфейсом оператора, удаленный терминал предоставляет несколько облегченную информацию об объекте управления (см. ниже).

Удаленный терминал выполнен по типу «тонкого клиента». На его экране видна работа апплета Java. Web-сервер, поставляющий данные удаленному терминалу для отображения, интегрирован в систему ЧПУ. Программное обеспечение терминала имеет модульную структуру, причем сами модули реализованы в виде библиотеки.

2.5.3. Библиотеки классов Java, используемые при создании апплетов

В языке Java все классы производны от класса Object и организованы в библиотеки. Библиотеки классов делятся на встроенные, подключаемые автоматически, например java.lang, и внешние, подключаемые с помощью оператора import.

На рис. 57 показана схема подключения библиотек, используемых при создании удаленного терминала. Библиотека JavaNcsCL поддерживает базовые функции работы с данными системы ЧПУ, такие как функции работающие с программируемым контроллером, управляющие формообразованием, отслеживающие статус технологического процесса и системы управления в целом. Библиотеку java.awt (Abstract Window Toolkit, инст-

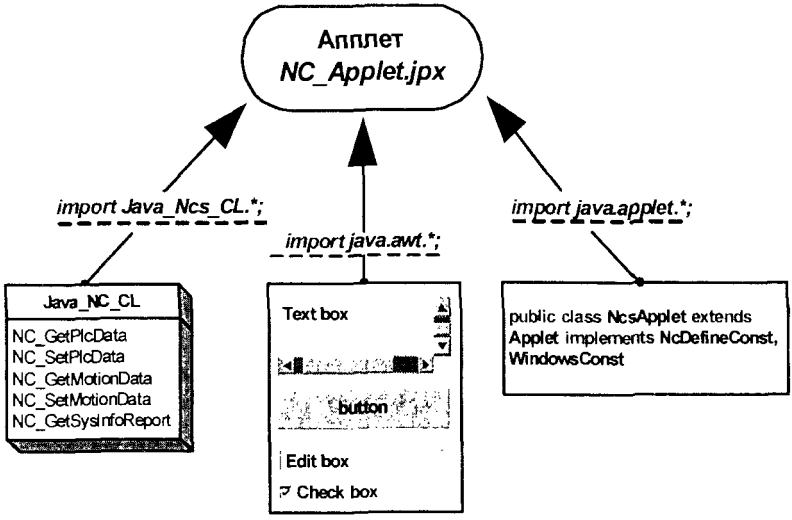


Рис. 57. Схема взаимодействия библиотек с апплетом Java

рументарий абстрактных окон) использовали для создания интерфейса оператора; с классами этой библиотеки работают апплеты Java. С ее помощью можно создавать обычные окна и диалоговые панели, кнопки, переключатели, списки, меню, полосы просмотра, одно- и многострочные поля для ввода текстовой информации. Управляющие элементы библиотеки позволяют создавать интерфейс оператора, не задумываясь о платформе, на которой выполняется апплет Java. Библиотека классов java.applet инкапсулирует базовое поведение апплетов Java. При создании апплета необходимо унаследовать его от класса Applet из библиотеки java.applet. Эта библиотека устанавливает также интерфейсы для подключения апплетов к их документам и классы для работы с мультимедиа.

Мы определились в выборе языка Java при реализации удаленного терминала, но теперь возникает проблема взаимодействия с ядром системы ЧПУ, написанном на C или C++, через TCP/IP канал. Данные, передаваемые в канале, ориентированы на C/C++ приложения оператора. Библиотека Java_NC_CL содержит классы, обеспечивающие конвертирование структур данных в пространстве между удаленным терминалом Java и C/C++ ядром системы ЧПУ. Структура трехуровневой библиотеки приведена на рис. 58.

Уровень, отвечающий за связь с ядром системы ЧПУ, реализует классы сокетов и класс таймера в механизме «time-out». Классы SynchronSocket и AsynchronSocket, унаследованные от стандартного класса Socket, обеспечивают синхронный и асинхронный способы обращения к серверу на базе

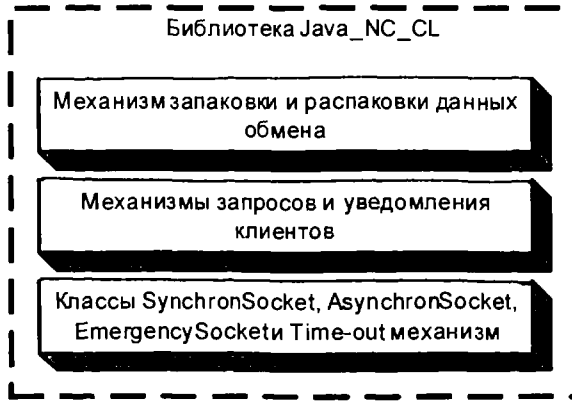


Рис. 58. Структура трехуровневой библиотеки классов

протокола TCP, а класс EmergencySocket, унаследованный от стандартного класса DatagramSocket, осуществляет контроль соединения на базе протокола UDP. Следующий уровень реализует базовые типы запросов (синхронный, асинхронный, по событию) и механизм уведомления зарегистрированных клиентов после обновления данных. Классы последнего уровня запаковывают отправляемые данные и распаковывают получаемые.

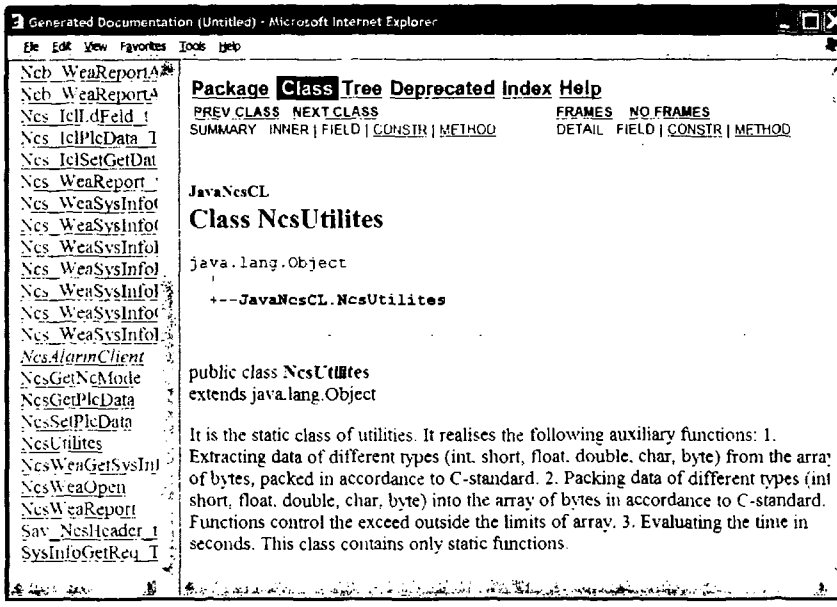


Рис. 59. Javadoc описание классов

Утилита Javadoc позволяет создавать интерактивный «help» с описаниями классов в формате HTML и методов (рис. 59), что помогает разрабатывать апплеты удаленного терминала.

2.5.4. Инструментарий разработки удаленного терминала

Разработка программного обеспечения удаленного терминала наиболее эффективна при комбинации CASE-системы (Computer-Aided Software Engineering) Rational Rose (фирмы Rational) со средой JBuilder (фирмы Borland). С помощью Rational Rose осуществляют проектирование удаленного терминала, построение диаграммы классов, состояния, взаимодействия (рис. 60), реинжиниринг модели из исходного кода [37] и т.д.Среда JBuilder предлагает профессиональные средства разработки, такие как: набор «Wizards» для создания каркасов приложений, интегрированный браузер Application Browser для управления проектом, визуальные дизайнеры, поддерживающие drag-and-drop механизмы, графический отладчик, высокоскоростной компилятор, системы визуализации UML-кода и управления конфигурациями, средства тестирования приложений и т.д. (рис. 61).

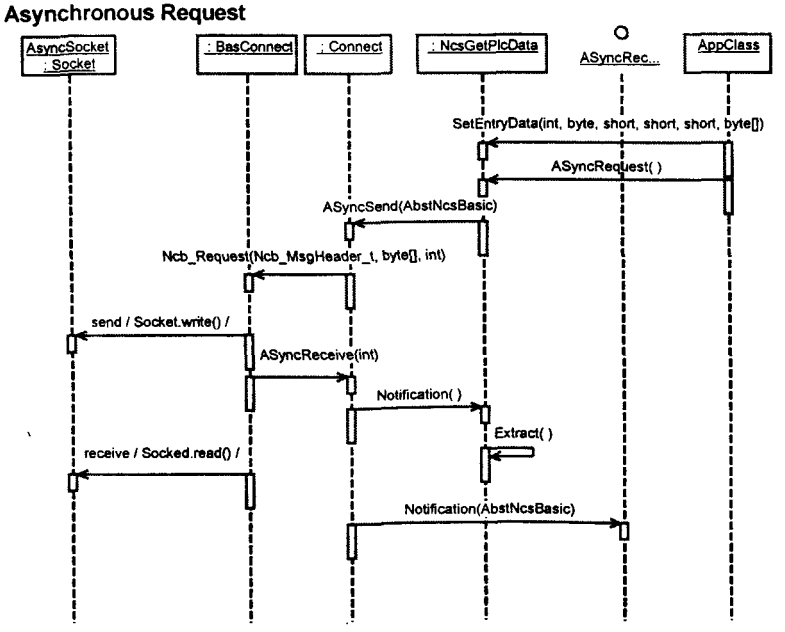


Рис. 60. Диаграмма взаимодействия при асинхронном запросе

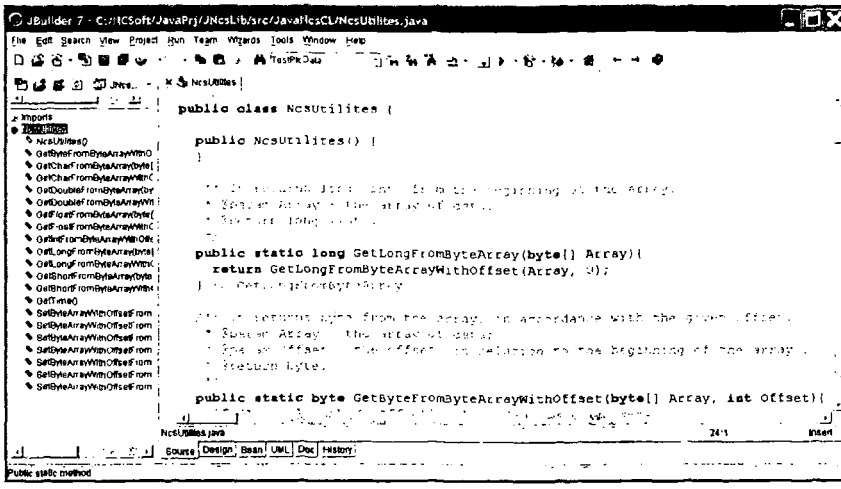


Рис. 61. Среда разработки удаленного терминала

2.5.5. Специфика удаленного терминала системы управления

Нормальный объем обменных данных терминала и ядра системы ЧПУ измеряется многими сотнями типов [38, 39]. Транзакции такого объема для удаленного терминала недопустимы в силу ограничений по быстродействию. Поэтому необходимо тщательно отбирать данные, которые удаленный терминал будет отображать, и сам способ такого отображения. Выделение основной информации из полного объема данных позволяет оптимизировать трафик.

Очень важно удачно подобрать управляющие элементы, осуществляющие вывод информации. Вариант Java NC_Applet представлен на рис. 62. В окне Connection выведена информация о системе ЧПУ WinPCNC (нашей разработки), с которой удаленный терминал поддерживает связь. В окно System Info поступают сообщения со стороны системы управления, включая сообщения об ошибках. В окна Current Position и End Position выводятся значения текущих и запрограммированных координат. Остальные окна использованы для работы с программируемым контроллером электроавтоматики. Стрелки связывают управляющие элементы с классами, поставляющими информацию из ядра системы ЧПУ.

Заключение

Создание удаленного терминала компьютерных систем управления стало возможным при использовании таких ресурсов Интернет-технологий, как «тонкий клиент/сервер» и апплеты Java. Ускорение процесса разра-

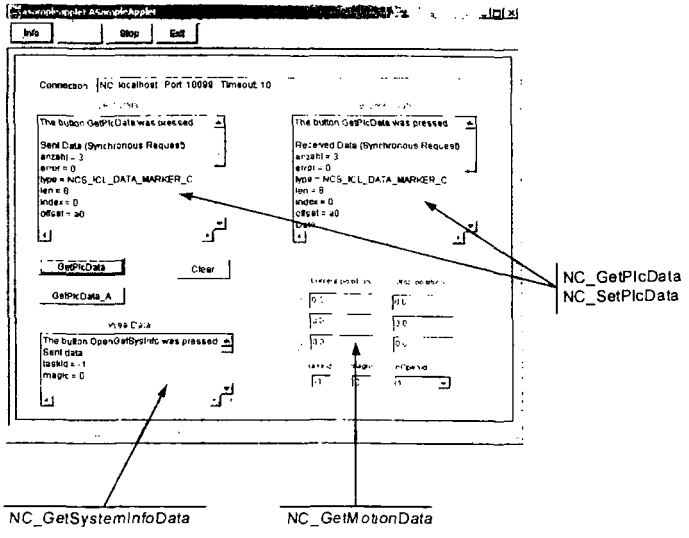


Рис. 62. Схема расположения функций, предоставляющих данные для управляющих элементов

ботки достигается за счет инструментальных средств проектирования, разработки, документирования и отладки программного обеспечения удаленного терминала.

Оптимизация трафика между ядром системы ЧПУ и удаленным терминалом предполагает тщательный отбор отслеживаемых данных. Некоторые информативные ограничения удаленных терминалов компенсируются исключительным удобством их применения.

2.6. Особенности архитектуры систем ЧПУ, поддерживающих стандарт ISO 14649 STEP-NC

Стандарт STEP используют для создания информационной модели изделия, работающей на всех этапах его жизненного цикла. Этапы перехода от системы автоматизированного проектирования CAD к системе автоматизированного программирования CAM достаточно хорошо согласованы. Однако внедрение заключительного этапа STEP-NC согласно стандарту ISO 14649 происходит достаточно сложно; более того, многие специалисты высказывают серьезные опасения, что этот стандарт будет отвергнут реальным производством. Сама идея STEP без ее заключительной фазы во многом теряет свой смысл. В этой связи в

данном разделе изложен вариант решения проблемы с использованием новейших информационных технологий.

Среди возможных видов интеграции в автоматизированных производствах в последнее время привлекают те, которые построены на единой информационной модели изделия в рамках его жизненного цикла: от компьютерного проектирования (CAD) и компьютерного планирования (CAPP) к автоматизированной подготовке управляющих программ (CAM) и изготовлению на станках с ЧПУ (NC). Подобная модель определена в рамках комплекса стандартов STEP. Слабым звеном в последовательных переходах по этапам жизненного цикла является переход CAM-NC, уверенное представление о котором не сложилось до сих пор. По этой причине акцент сделан именно на этом переходе, причем речь далее пойдет о той части стандартов STEP, которая определена для области обработки резанием на станках с ЧПУ [40-49].

2.6.1. Традиционное программирование станков с ЧПУ и стандарт STEP-NC

Программирование современных систем ЧПУ подчиняется стандарту ISO 6983 (DIN 66025), которому уже более 50 лет и который явно тормозит развитие ЧПУ-технологии. Стандарт поддерживает простые команды для элементарных перемещений и логических операций, но не сложные геометрию и логику. Управляющие программы в стандарте ISO 6983 содержат ничтожное подмножество информации, полученной на уровне CAD-CAM систем. Однако более серьезным является невозможность двустороннего обмена информацией с этими системами. Это означает, что любые изменения в управляющей программе не могут быть отображены в восходящем информационном потоке к системам CAD-CAM [50] (рис. 63).

В отличие от существующей ситуации стандарт STEP-NC ISO 14649 предлагает модель того, *что* нужно сделать на уровне системы ЧПУ, но не подробности того, *как* осуществлять траекторные перемещения и выполнять команды логических переключений. Это определяет специальную структуру управляющей программы ЧПУ (*program structure*), которую используют для построения логических блоков в рамках структурного программирования обработки.

Структура управляющей программы не является списком типовых обрабатываемых форм (*features*) (см. ниже); она определяет план операции (*workplan*), который представляет собой последовательность исполняемых объектов (*executables*). Кроме того, возможны: свободная организация процесса обработки (*non-sequential*), параллельные структуры (*parallel*), циклы (*while-statement*), условные переходы (*if-statement*) и др. Исполняемые объекты (*executable*) в составе плана операции инициируют активность

Рис. 63. Структура двустороннего потока информации между инженерными службами и цеховым уровнем

станка. Существуют три типа исполняемых объектов: собственно план операции (*workplan*), функция ЧПУ (*NC function*), шаг операции (*workingstep*) (рис. 64).

Шаг операции *workingstep* описывает процессы, в которые вовлечены интерполируемые координатные оси. В отличие от этого функции ЧПУ (*NC function*) сопоставлены единичным событиям и с интерполяцией не связаны. Шаг операции *workingstep* является важнейшим строительным блоком управляющей программы ЧПУ стандарта STEP-NC ISO 14649. Блоки могут быть нейтральными действиями – ускоренными перемещениями *rapid movement*, измерительными циклами *touch probing*, а также технологическими шагами операции *machining workingstep*. Реальное содержание шага операции *workingstep* специфицировано в объекте-переходе *operation*. Существует возможность повторного использования информации перехода *operation* (но не *workingstep*) для нескольких типовых форм обработки *features*. Итак, переход *operation* может быть ассоциирован со многими типовыми формами *features* и использован в разных местах. С другой сто-

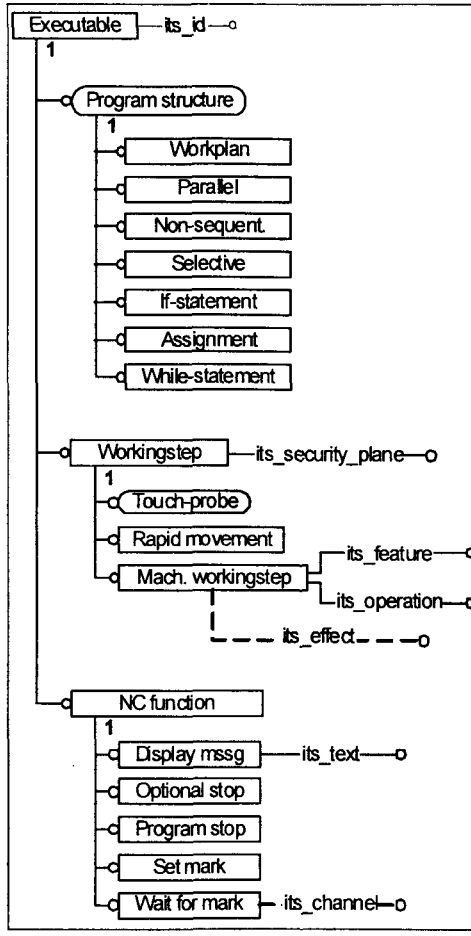


Рис. 64. Компоненты управляющей программы ЧПУ соответственно стандарту ISO 14649 в виде набора исполняемых объектов «executables»

местах. С другой стороны, шаг операции *workingstep* уникален. Дублирование этого шага в пределах плана операции *workplan* в точности воспроизведет те же самые действия станка. Переход содержит технологический алгоритм (включая стратегию внедрения в материал и вывода инструмента) и указания по настройкам. Переходы имеют черновую и чистовую версии. Предполагается, что интеллектуальные системы ЧПУ будут самостоятельно рассчитывать траектории инструмента для стандартных типовых форм. Переходы *operations*, согласно стандарту ISO 14649, находятся на самом нижнем информационном уровне процесса управления обработкой.

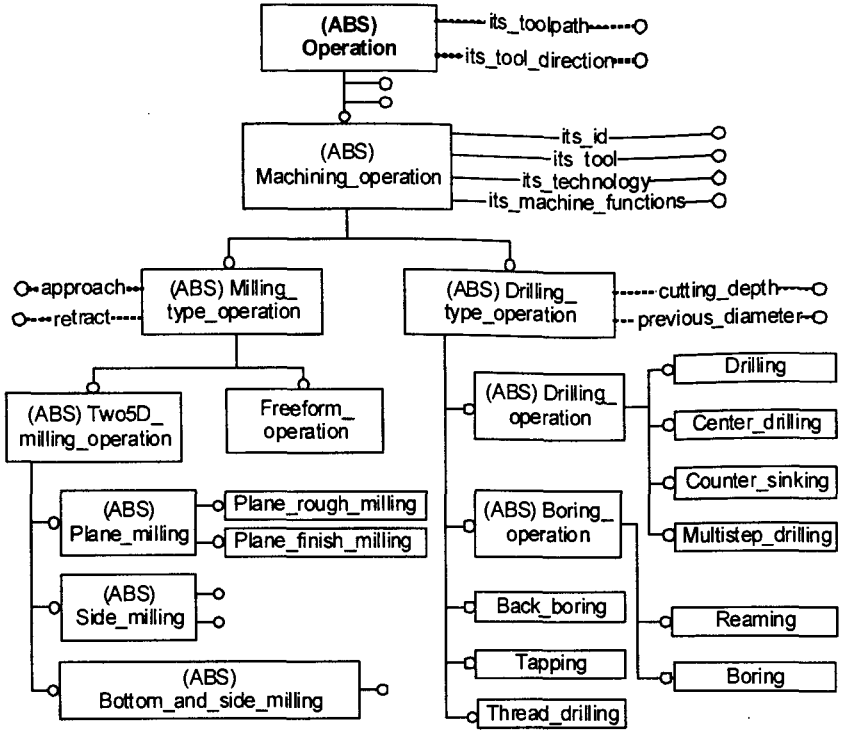


Рис. 65. Структура перехода: (ABS) – абстрактный объект; Two5D – обработка типа 2,5D

Помимо прочего, там описана траектория инструмента, если того требует CAM-система или система ЧПУ. Структура перехода показана на рис. 65.

2.6.2. Язык EXPRESS

Описанные выше компоненты управляющей программы ЧПУ являются объектами данных *entities*, которые могут быть формально представлены на языке EXPRESS. Далее приведены примеры такого представления для структуры управляющей программы *program_structure*, для технологического шага операции *machining_workingstep* и плана операции *workplan*.

```
ENTITY program_structure
ABSTRACT SUPERTYPE OF (ONE OF(workplan, parallel,
non_sequential, selective, if_statement, while_statement,
assignment))
SUBTYPE OF (executable);
END_ENTITY;
```

```
ENTITY machining_workingstep
SUBTYPE OF (workingstep);
its_feature: manufacturing_feature;
its_operation: machining_operation;
its_effect: OPTIONAL in_process_geometry;
END_ENTITY;
```

Рассмотрим поля объекта технологического шага операции:

- its_feature: типовая форма обработки, с которой работает технологический шаг операции;
- its_operation: переход;
- its_effect: изменение геометрии детали в результате выполнения перехода. CAM-система может использовать этот атрибут для предсказания эффекта перехода на геометрию детали, а система ЧПУ может сравнить предсказанные изменения с теми, которые обусловлены ее внутренними алгоритмами.

```
ENTITY workplan
SUBTYPE OF (program_structure);
its_elements: LIST[1:?] OF executable;
its_channel: OPTIONAL channel;
its_setup: OPTIONAL setup;
its_effect: OPTIONAL in_process_geometry;
WHERE
WR1: SIZEOF(QUERY(it <* its_elements | it = SELF)) = 0;
END_ENTITY;
```

Поля плана операции *workplan*:

- its_elements: последовательность исполняемых объектов *executables*;
- its_channel: идентификатор канала, используемого для выполнения плана операции *workplan* (только для тех систем ЧПУ, которые поддерживают многоканальное управление);
- its_setup: настройка, включающая определение для плана операции глобальной безопасной плоскости *security plane* и смещений нуля, к которым обращаются все *features*;
- its_effect: изменение геометрии заготовки в результате перехода *operation*.

Язык EXPRESS имеет и графическую нотацию, которая использована на рис. 64, 65 и др.

2.6.3. Процессы и ресурсы в STEP-NC

Управляющая программа, являясь процессом, работает в окружении ресурсов. Упрощенная схема процессов и ресурсов при обработке изделия взаимодействия показана на рис. 66.

Цель организации рабочего процесса состоит в обработке типовых форм «features» с целью получения готового изделия *workpiece*. При этом используются методы, имеющие отношение к типовым формам и описанию изделия в целом. Цель достигается на основе использования плана операции и его шагов, операций, технологической стратегии, траекторий инструмента.

Изделие получают из заготовки удалением типовых форм *features*. Это делается путем условного или безусловного выполнения ассоциированных с типовыми формами шагов операции *workingsteps* в потоке управления, задаваемом исполняемыми блоками *executables*. При этом соблюдаются необходимые допуски и используется инструмент, отвечающий всем необходимым требованиям. Такая модель использует информацию STEP-форматов проектирования изделия, прикладные протоколы AP204 и AP213 (Application Protocol) вплоть до этапа интерпретации управляющей программы, т.е. она несопоставимо богаче существующей схемы программирования. Предполагается, что система управления способна интерпрети-

Рис. 66. Взаимодействие процессов и ресурсов при обработке изделия

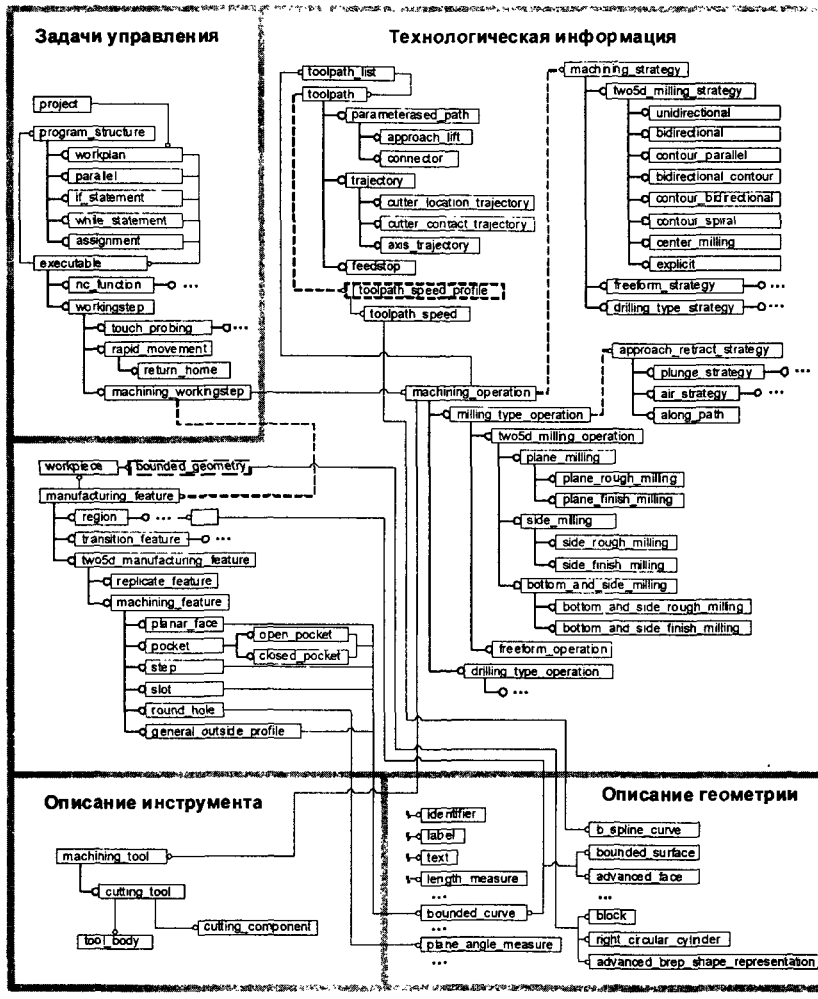


Рис. 67. Информация стандарта ISO 14649 на языке EXPRESS-G

ровать подобную информацию и генерировать необходимые перемещени и циклы.

Стандарт ISO 14649 предоставляет системе ЧПУ обширную связанну информацию (рис. 67), которая состоит из четырех разделов: описания з дач управления, технологической информации, описания инструмента геометрического описания. Раздел задач представляет собой логическу последовательность исполняемых блоков и типов данных. Детали кажд го шага операции описаны в разделе технологической информации, пр

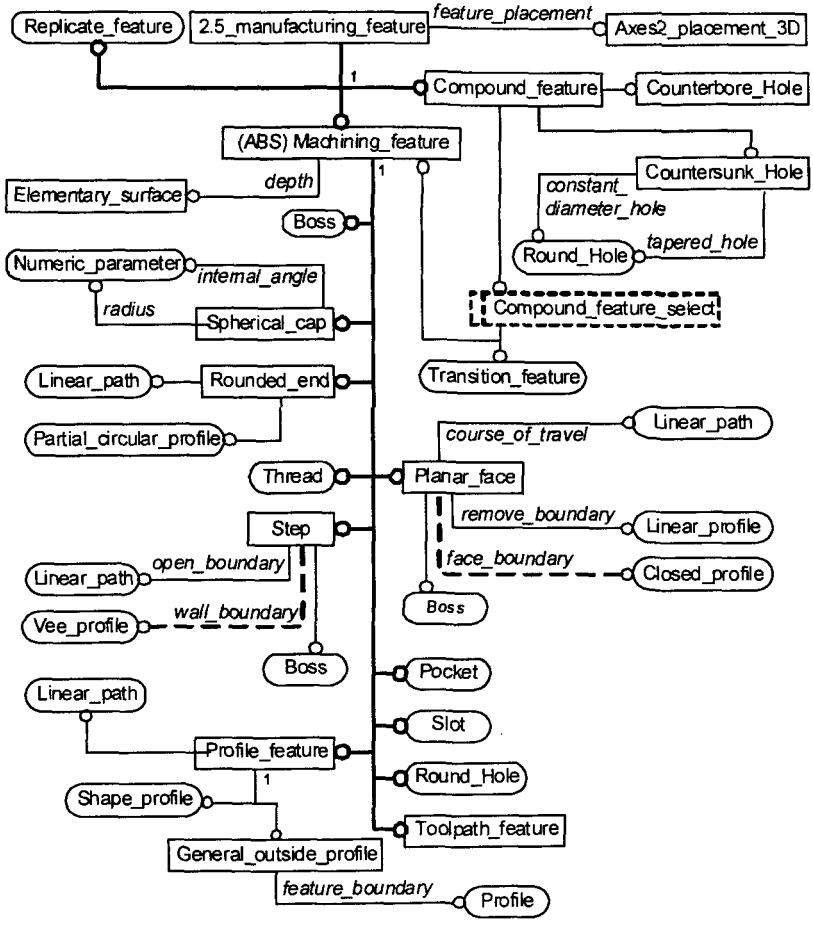


Рис. 68. Структура типовой формы «feature»

чем они связаны отношениями с описанием инструмента и геометрическим описанием.

Важнейшим элементом технологического процесса служат типовые формы *features*, которые определяют области удаляемого материала заготовки, а их внешний вид является частью внешнего вида изделия *workpiece*. Типовые формы задают параметрически или в виде совокупности образующей и направляющей. Особый случай представляют поверхности свободной формы, для которых определяют область, в пределах которой размещается поверхность свободной формы.

Далее на языке EXPRESS представлено формальное описание объекта-изделия *workpiece* и типовой объектной формы *features*.

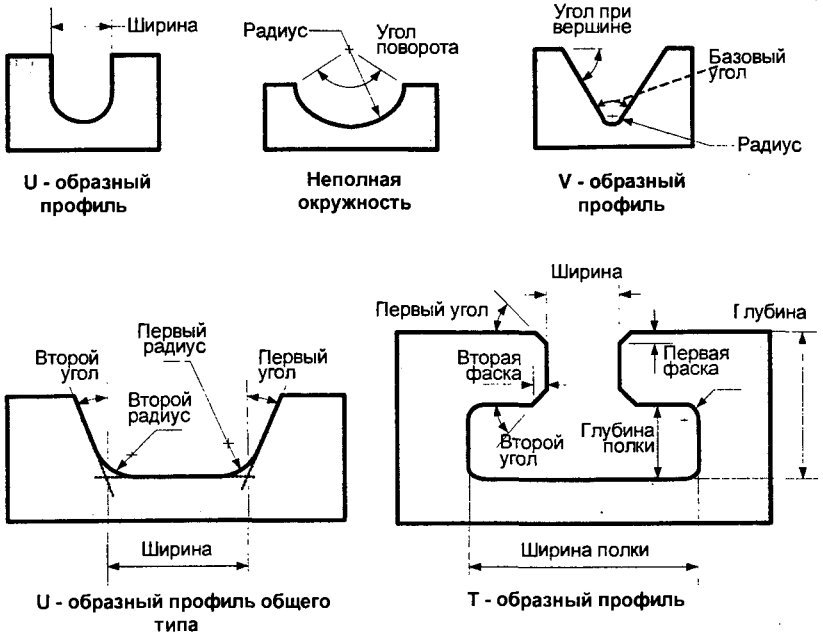


Рис. 69. Примеры типовых форм при фрезерной обработке

```
ENTITY workpiece;
its_id: identifier;
its_material: OPTIONAL material;
global_tolerance: OPTIONAL shape_tolerance;
its_rawpiece: OPTIONAL workpiece;
its_geometry: OPTIONAL advanced_brep_shape_representation;
its_bounding_geometry: OPTIONAL bounding_geometry_select;
clamping_positions: SET [0:?] OF cartesian_point;
END_ENTITY;
```

- its_id: уникальный идентификатор изделия;
- its_material: этот атрибут идентифицирует материал изделия. Он также используется при определении параметров технологического процесса обработки;
- global_tolerance: допуск на изготовление изделия; он действует там, где не указаны другие допуски;
- its_rawpiece: здесь может быть указана геометрия заготовки;
- its_geometry: точное описание геометрии конечного изделия соответственно стандарту ISO 10303-514;
- its_bounding_geometry: с помощью этого атрибута пограничная геометрия изделия может быть представлена призмой, цилиндром или

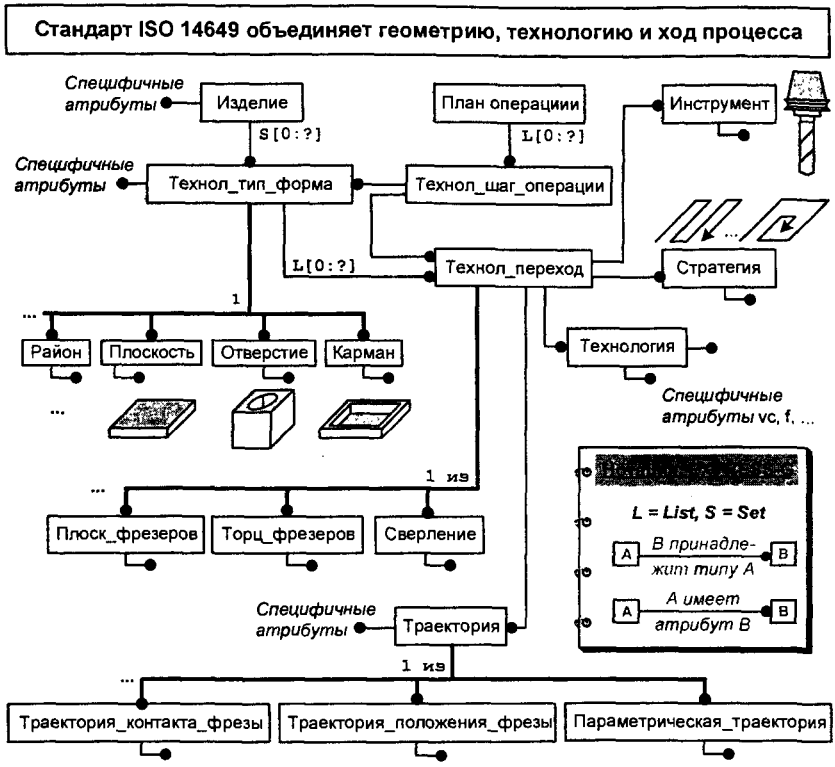


Рис. 70. Система понятий на русском языке стандарта ISO 14649 в нотации EXPRESS-G

точным геометрическим описанием соответственно объекту advanced_brep_shape_representation в стандарте ISO 10303—514 (brep – bounding representation);

- clamping_positions: положения зажимных устройств на поверхности изделия.

Типовую форму *feature* описывают с помощью ее геометрических свойств, при этом отсутствуют какие-либо указания о способах обработки типовой формы. Подобная информация содержится только в переходах *operations* и определяется технологическими алгоритмами-методами. Методы рассмотрены в разделе геометрической информации стандарта ISO 14649.

ENTITY manufacturing_feature

ABSTRACT SUPERTYPE OF ONE OF(region,two5D_manufacturing_feature, transition_feature));

its_id: identifier;

its_workpiece: workpiece;
its_operations: SET [0:?] OF machining_operation;
END_ENTITY;

- its_id: каждая типовая форма имеет уникальный идентификатор;
- its_workpiece: изделие, частью которого является типовая форма;
- its_operations: набор переходов, ассоциированных с типовой формой, необходимых для ее обработки. Необязательно, чтобы переходы выполнялись последовательно один за другим – этот порядок определяется планом операции *workplan*. Точно так же, необязательна жесткая последовательность обработки типовых форм, которая скорее определяется критерием минимизации смены инструмента.

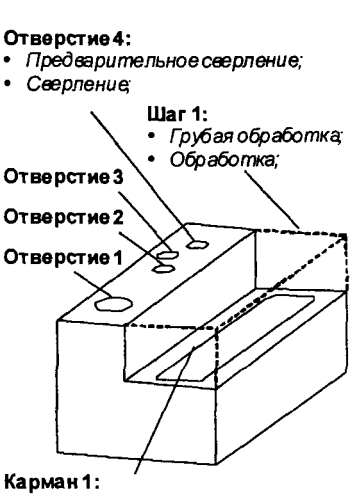
Структура объектной типовой формы *feature* показана на рис. 68. Примеры чертежей конкретных типовых форм *features* приведены на рис. 69 в иллюстративных целях. Эти формы являются извлечением из полного архива, разработанного в рамках стандарта ISO 14649. Еще раз (сравните с рис. 67) обратимся к тесному взаимодействию задач управления, технологической информации, инструмента и геометрической информации, но на этот раз в лаконичной упрощенной и более ясной форме (рис. 70). Наша цель состоит в осмыслении понятийного аппарата стандарта ISO 14649 и использовании соответствий на русском языке.

2.6.4. Смешанная архитектура

До сих пор станки с ЧПУ программируют в стандарте ISO 6983. Этот стандарт существует со времени использования перфолент и перфокарт, он абсолютно не удовлетворяет современным технологиям. Управляющие программы, соответствующие ISO 6983, всего лишь описывают координатные перемещения (G1, G2, G3) и управляют циклами (M3, M8). Новые языки программирования работают с технологическими задачами, привязанными к типовым формам (features). Фрагмент управляющей программы с использованием типовых форм приведен на рис. 71.

Все операции, необходимые для перехода от заготовки к готовому изделию, могут быть описаны в терминах технологических задач. В этой связи на цеховой уровень поступает огромный объем информации. Все модификации цехового уровня могут быть не только сохранены, но и без труда переданы обратно в отделы планирования.

Поскольку геометрия и заготовки, и готового изделия описывается с использованием STEP-синтаксиса, возможен прямой обмен информацией между CAD/CAM/CNC системами. Геометрические данные могут быть непосредственно импортированы в систему ЧПУ, при этом должна быть добавлена технологическая информация, чтобы сгенерировать управляющую программу.



Изделие, полученное в результате 2.5D-обработки, описано с помощью типовых форм, а технология представлена шагами операций

```
// header
ISO- 10303- 21
HEADER;
ENDSEC;

DATA;
// workpiece and work plan
#1=WORKPIECE(...);
#2=MATERIAL(...);
#3=WORKPLAN("name",(#10, #11,...),...);

// working steps
#10=MACHININD_WORKINGSTEP("hole1",#20,
                                ....);
#11=MACHININD_WORKINGSTEP(
                    "Pocket1_plunge",....);
...

// manufacturing features
#20=ROUND_HOLE(...,#1, #30, #40,...);
...

// geometric data
#30=CARTESIAN_POINT(...);
...

// operation data
#40=DRILLING(..., #50,);
#41=BOTTOM_AND_SIDE_ROUGH_MILLING(...);
...

// tool data
#50=CUTTING_TOOL("spiral_drill_9mm"...);
```

Рис. 71. Фрагмент управляющей программы с использованием типовых форм

Рассматривая структуру системы ЧПУ, ориентированную на использование STEP-NC, следует заметить, что в течение продолжительного времени будут существовать смешанные варианты, способные воспринимать управляющие программы в стандарте ISO 6983. В этот переходный период от ISO 14649 к ISO 6983 и CAM-системы, и системы ЧПУ будут вынуждены поддерживать оба стандарта (рис. 72). Стандарт STEP-NC будет иметь более высокий приоритет.

Представленный прототип системы ЧПУ воспринимает данные из нескольких источников: от CAD-CAM системы, из библиотеки, через графический интерфейс, посредством ручного ввода данных. Комбинации типовых форм и их геометрических описаний в совокупности с технологической информацией порождают шаги операции. Геометрическая модель изделия построена на основе стандарта ISO 10303, AP203. Производственные данные содержат описания типовых форм, технологии и инструмента в стандартах ISO 10303, AP224 и AP214. Эти данные служат базисом для выбора типовых форм и шагов операции, результатом работы которых ста-

новится готовое изделие. Последовательность шагов операции определяет специфику рабочего процесса, инициируемого системой ЧПУ.

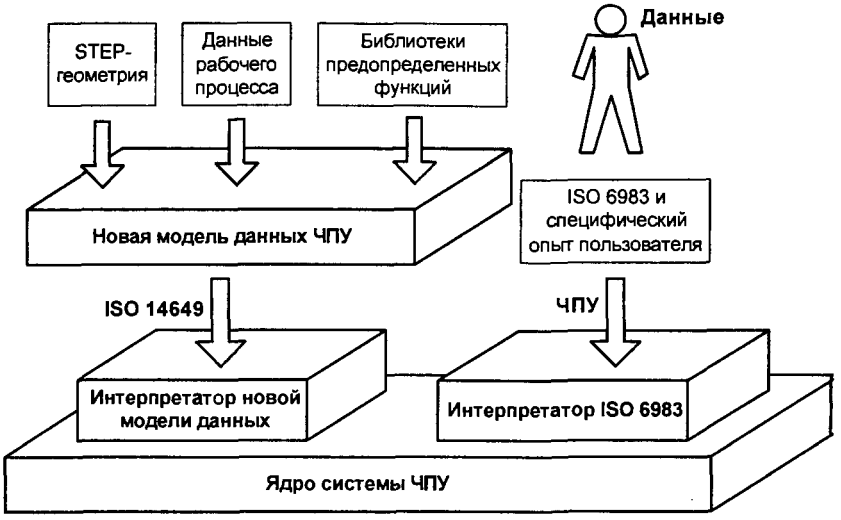


Рис. 72. Смешанная архитектура системы ЧПУ, поддерживающая стандарты ISO 14649 и ISO 6983

Глава 3.
Задачи управления

При переходе от архитектуры системы ЧПУ к ее математическому обеспечению (МО) необходимо сформировать некоторый подход к систематизации, который объяснил бы логику построения обеспечения, сделал бы процесс его разработки прозрачным и регулярным. Этот подход состоит в выделении задач ЧПУ в качестве автономных объектов разработки. В числе этих задач – геометрическая, логическая, терминальная и диагностическая. Тот или иной набор их зависит от конфигурации конкретной системы управления и особенностей объекта управления. Этот набор определяет, в конечном счете, функциональные возможности управления.

3.1. Реализация геометрической задачи

Рассмотрена геометрическая задача ЧПУ с такими ее важнейшими компонентами, как интерпретатор управляющих программ и интерполятор. Показано, как изменились требования к этим компонентам за последнее время: интерпретатор должен быть настраиваем на любую версию кода ISO-7bit управляющих программ; интерполятор должен иметь открытую (расширяемую) архитектуру и допускать любую комбинацию алгоритмов интерполяции. Приведен пример практической реализации компонентов геометрической задачи.

Математическое обеспечение системы ЧПУ на прикладном уровне состоит из нескольких фундаментальных разделов, называемых задачами ЧПУ [1, 20]. Важнейшей из таких задач является геометрическая (*motion control*), которая присутствует во всех без исключения системах ЧПУ типа PCNC. В свою очередь геометрическая задача состоит из трех крупных модулей: интерпретатора управляющих программ, интерполятора, модуля управления следящими приводами. Последний модуль сильно зависим от типа следящих приводов и способа замыкания позиционных контуров, в то время как для двух первых модулей могут быть предложены инвариантные решения. В этой связи остановимся на проблемах реализации двух первых модулей.

3.1.1. Интерпретатор управляющих программ

Интерпретатор транслирует кадры управляющей программы в коде ISO-7bit с целью представления данных во входном формате интерполятора. В фазе интерпретации кадра система ЧПУ выполняет эквидистантные расчеты и расчеты, связанные со стыковкой эквидистантных контуров; осуществляет преобразование координатных систем (в абсолютную или относительную системы) и преобразование систем измерения (в миллиметры или дюймы); вызывает стандартные циклы и подпрограммы; разделяет потоки данных геометрической, логической и других задач.

Наилучший вариант реализации интерпретатора состоит в его построении по типу ISO-процессора [51], поскольку такое решение обеспечивает наибольшее быстродействие и гибкость системы PCNC в отношении системы команд, т.е. версии языка (кода) ISO-7bit. С момента наших первых разработок архитектуры ISO-процессора прошло семь лет, за это время она претерпела существенные изменения и была многократно внедрена. Рассмотрим архитектуру ISO-процессора в ее настоящем (объектно-ориентированном) виде, с выделением независимого и зависимого уровней.

Выделение уровней в объектной реализации ISO-процессора позволило выделить программные компоненты, одинаковые для всех версий языка ISO-7bit. На независимом уровне обозначены блоки и разработаны соответствующие им классы объектов: конвертор внутреннего формата (класс CParcerConverer), прототип кадра (класс CBlock), прототипы геометрических перемещений (классы CPoint, CLine, CCircle), обобщенные прототипы интерпретируемого кадра (класс CAbstractSubProcessor, поддерживающий механизм работы с групповыми интерпретаторами и эквидистантные расчеты), вспомогательные модули и соответствующие вспомогательные классы для буферизации, форматирования данных и др. На зависимом уровне определены G-команды конкретной версии кода ISO-7bit и соответствующие классы, производные от обобщенного класса CGCommand, а также обозначены блоки и разработаны соответствующие им классы объектов: прототип интерпретируемого кадра с уточненным набором групповых интерпретаторов и координат G-вектора (класс CSubProcessor); прототип интерпретатора (класс CISOProcessor) с конкретной системой команд (класс CInst_GCommands). Полная объектная архитектура ISO-процессора показана на рис. 73.

Интерпретация кадров управляющей программы построена на основе конвейера и выполняется за семь шагов (рис. 74). На завершающей стадии данные поступают в кольцевой буфер, позволяющий анализировать на совместимость группу соседних кадров с эквидистантной коррекцией. Окончательный результат интерпретации представлен в виде IPD-кода (InterPolator Data). Содержательно последовательность шагов интерпретации состоит в следующем.

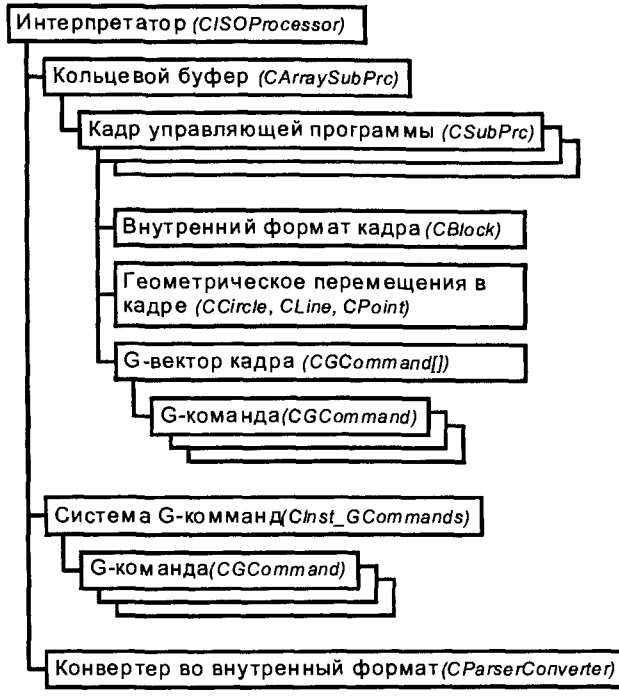


Рис. 73. Полная объектная архитектура ISO-процессора

Шаг 1: перевод (конвертирование) кадра управляющей программы во внутренний формат. В кадре выделяются G-функции (команды ISO-процессора), перемещения (первый тип параметров), адреса (второй тип параметров) и комментарии. Пример подобного «разбора» кадра показан в табл. 5.

Таблица 5. Пример разбора кадра на первом шаге интерпретации

Пример кадра	N20	G91	G01	X20.5	Y37.5	F2500	*Comment
G-функции		G91	G01				
Перемещения				X20.5	Y37.5		
Адреса	N20					F2500	
Комментарии							*Comment

Шаг 2: формирование активного G-вектора, размерность (число координат) которого соответствует числу групп G-функций (команд) в рабочей версии кода ISO-7bit. Всякая новая G-команда включается в ту координату

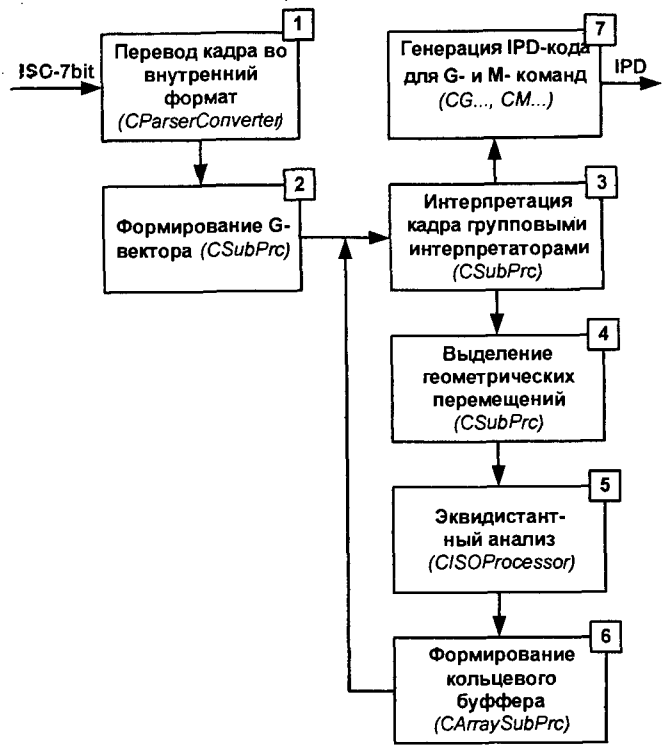


Рис. 74. Конвейер ISO-процессора

G-вектора, которая соответствует номеру группы G-команды, и будет существовать, пока ее не заменит другая модальная G-команда из той же группы. Немодальные команды деактивируются в G-векторы после завершения интерпретации кадра.

Шаг 3: интерпретация кадра групповыми интерпретаторами (Dimension, Plane, Delay и др.), которые привязаны к группам G-функций и, соответственно, координатам G-вектора. Вызов того или иного группового интерпретатора определяет порядок выполнения интерпретации, который непосредственно зависит от версии кода ISO-7bit.

Шаг 4: назначение геометрических перемещений (при активной эквидистантной коррекции) в объектах классов CLine, CCircle, CPoint.

Шаг 5: эквидистантная коррекция, в процессе которой осуществляется перерасчет траектории инструмента с учетом его размеров.

Шаг 6: формирование кольцевого буфера кадров и «окна просмотра траектории». Цель состоит в корректной стыковке соседних эквидистантных контуров.

Шаг 7: генерация выходного IPD-кода активными командами G-вектора в соответствии с текущими значениями параметров после обращения к ним групповых интерпретаторов.

Объектная реализация архитектуры и системы команд ISO-процессора насчитывает более 70 классов, поэтому поясним далее лишь некоторые моменты. Диаграмма конвейера интерпретатора представлена на рис. 75. Основной цикл создает функция ExecuteBlock() класса CISOProcessor – она вызывает функцию CalckGVectorBlock() того же класса для выполнения двух первых шагов конвейера. После этого вызывается функция Interpretator() класса CSubPRC (текущего интерпретируемого кадра) для выполнения третьего шага, в рамках которого и вызываются групповые интерпретаторы StandartCycles(), Debug(), Dimension(), Plane(). Если эквидистантная коррекция активна, то вызывается функция Equidistant_Task()

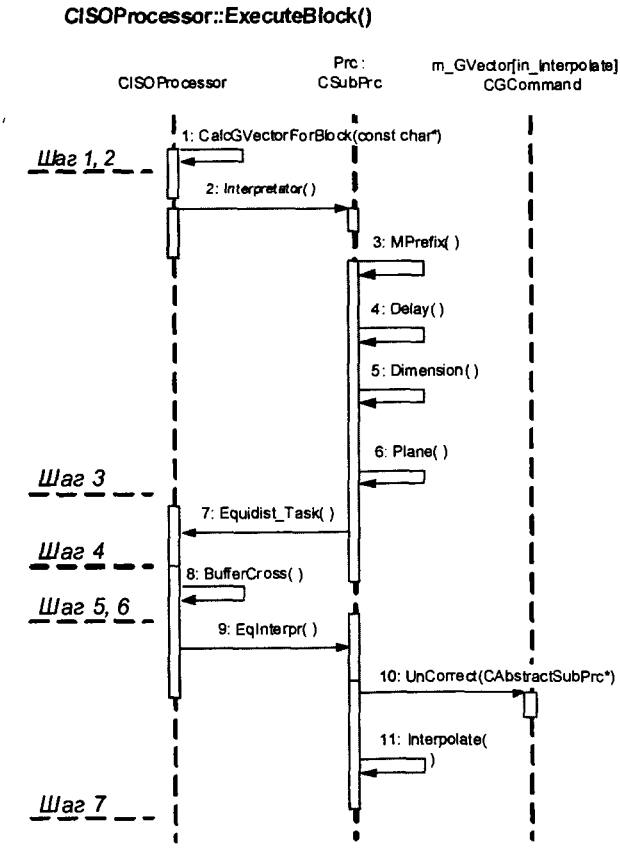


Рис. 75. Диаграмма конвейера ISO-процессора

класса CISOProcessor для выполнения четвертого шага. Функция BufferCross() реализует пятый-шестой шаги, а остальные функции – седьмой шаг. В качестве примера на рис. 76 показана диаграмма объектов, построенная для двух первых шагов конвейера интерпретатора.

На первом шаге конвейера, реализующем функцию CalcGVectorForBlock(), вызывается Identifier() конвертора для синтаксического анализа очередного кадра управляющей программы. В процессе конвертации кадра в формат команд интерпретатора осуществляется инициализация Init() внутреннего представления. После этого кадр формата ISO-7bit преобразуется с помощью функции Unification(), в некоторую стандартную форму, при этом расставляются или убираются пробелы, меняются регистры и т.д. Вслед за вызовом функций SetGFunc(), SetMFunc(), SetAxis(), SetAddress() класса CBlock выполняется разбор кадра аналогично примеру в табл. 5. Функция Activate() класса CISOProcessor реализует второй шаг конвейера. Рисунок 76 демонстрирует использование широко известной CASE-системы RationalRose при проектировании и объектной реализации ISO-процессора.

На рис. 77 показан пример практической реализации ISO-процессора в рамках двухкомпьютерной системы ЧПУ типа PCNC.

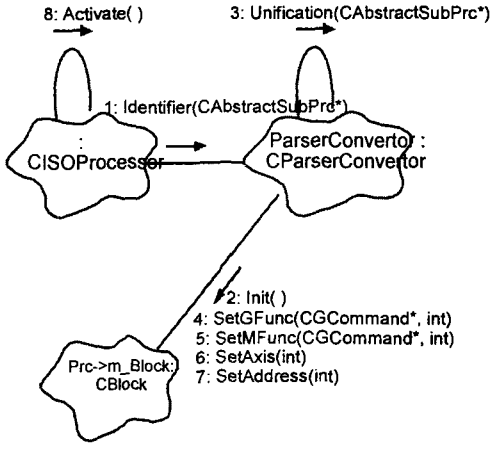


Рис.76. Фрагмент диаграммы объектов конвейера

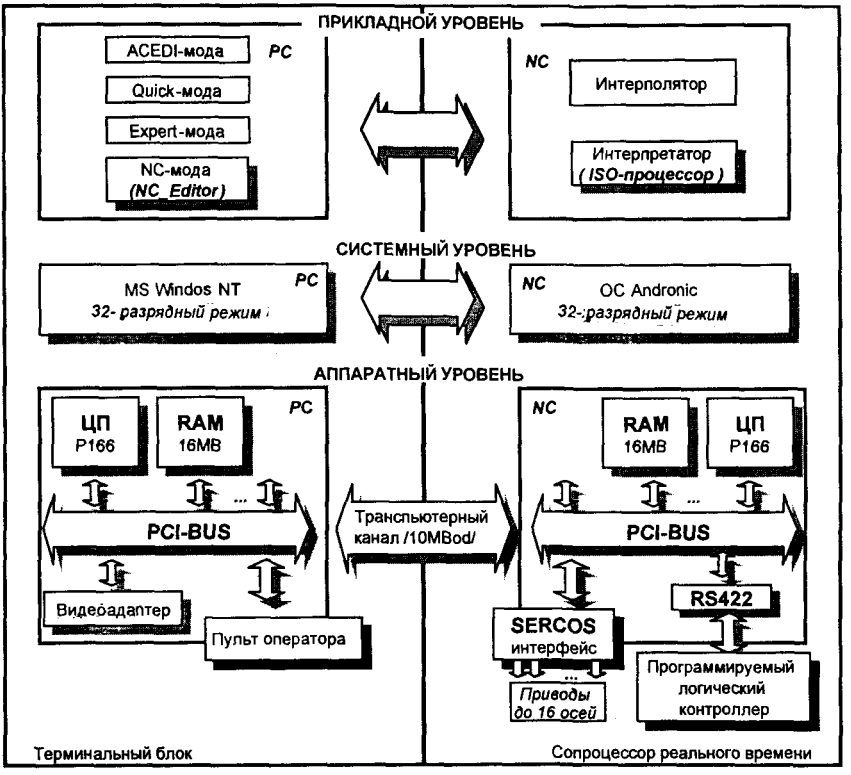


Рис. 77. Интеграция ISO-процессора в двухкомпьютерную систему ЧПУ типа PCNC

3.1.2. Интерполятор

Задачи модуля интерполяции полагаются традиционными, однако в последнее время к интерполятору предъявляют новые требования. В их числе: уменьшение цены дискреты в приводе до 0,5 микрона и меньше; прямой выход на приводы, при котором перемещение в кадре задано в приращениях следящего привода, что необходимо при особо высоких скоростях подачи; разложение сложных перемещений на линейные комбинации основных перемещений. Подобные требования определяют новую (открытую) архитектуру интерполятора, в которой четко обозначены отдельные блоки.

Открытый интерполятор допускает свободное наращивание алгоритмов интерполяции и произвольную их комбинацию при воспроизведении сложных траекторий в многокоординатном пространстве (в том числе и с использованием сплайнов). Ключевым моментом при построении открытого интерполятора является удачный выбор входных форматов. Далее при-

веден фрагмент формального грамматического описания одной из версий входного формата, в составе которого управляющие структуры (заголовки, HeaderLine) и данные (DataLine).

<заголовок> - <старт интерполяции, подача задана явно>|
<старт интерполяции, подача задана неявно>|<вызов подпрограммы>|
<конец программы>|<комбинация алгоритмов>|
<переопределение скорости подачи>|<относительный номер кадра>;
<старт интерполяции, подача задана явно> - HL_i01(<идентификатор осей>,<подача в мм/мин>);
<старт интерполяции, подача задана неявно> - HL_i02(<идентификаторы осей>, <индекс подачи>);
<вызов подпрограммы> - HL_i04(<размер подпрограммы в байтах>);
<конец программы> - HL_End;
<комбинация алгоритмов> - HL_n00(<маска интерполяции>);
<маска интерполяции, 16-разрядное слово> - <признак линейной>
<признак круговой><признак сплайновой><признак полиномиальной>
<признак алгоритма Безье><признак алгоритма Рябенкова>
<резервные биты>;
<переопределение скорости подачи> - HL_n01(<новая подача>);
<данные> - <линейная интерполяция>|<сплайновая интерполяция, заданы углы входа и выхода>|<сплайновая интерполяция, задан угол выхода>|<круговая интерполяция>|<переопределение плоскости>|
<переопределение осей>;
<линейная интерполяция> - DL_06
(<относительные координаты прямой>);
<сплайновая интерполяция, заданы углы входа и выхода>- DL_07(<относительные координаты точек>,<углы входа-выхода>);
<сплайновая интерполяция, задан угол выхода> - DL_08(<относительные координаты точек>,<угол выхода>);
<круговая интерполяция> - DL_11(<относительные координаты для окружности>);
<переопределение плоскости> - DL_16(<код плоскости>);
<переопределение осей> - DL_17(<коды осей>).

Заголовок HL_i01 инициирует старт (начало интерполяции или включение быстрого хода), при этом работают алгоритмы разгона или торможения. Переопределение осей HL_n01 производится, если задействованные в текущем кадре оси интерполяции отличаются от ранее установленных. Каждый кадр генерирует HL_n05 со своим относительным номером. Кадры, не содержащие информации для интерполятора, только увеличивают относительный номер следующего кадра. Признак «i» в заголовке означает прерывание интерполяции, а признак «n» – отсутствие прерывания.

В табл. 6. приведены примеры исходных текстов в коде ISO-7bit и соответствующие IPD-форматы на входе в интерполятор.

Таблица 6. Результаты работы ISO-процессора по формированию IPD-форматов

№	Программа	IPD-форматы
1	N10 %CNC-Test1	HL_n05(0)
	N20 *comment	
	N30 G90	
	N40 G00 X20 Y20	HL_n05(3) HL_i02(10,0) DL_06(20,20)
	N50 G01 X50 Y50	HL_n05(1) HL_i02(10,1) DL_06(30,30)
	N60 X100	HL_n05(1) DL_17(8) DL_06(50)
	N70 Y100	HL_n05(1) DL_17(2) DL_06(50)
	N80 M30	HL_n05(1) HL_End
2	N10 %CNC-Test2	HL_n05(0)
	N20 *comment	
	N30 G91 G00 X20 Y20	HL_n05(2) HL_i02(10,0) DL_06(20,20)
	N40 G01 X30 Y10 F2000	HL_n05(1) HL_i01(10,2000) DL_06(30,10)
	N50 X100	HL_n05(1) DL_17(8) DL_06(100)
	N60 Y100	HL_n05(1) DL_17(2) DL_06(100)
	N70 G02 X10 Y10 I10 J0 F2500	HL_n05(1) HL_n01(2500) DL_17(10)
		DL_11(...)
	N80 M30	HL_n05(1) HL_End
3	N10 %CNC-Test3	HL_n05(0)
	N20 *comment	
	N30 G91	
	N40 G00 X30 Y30	HL_n05(3) HL_i02(10,0) DL_06(30,30)
	N50 G300 G01 Z50	HL_n05(1) HL_i01(12,2500) HL_n00(3)
		DL_17(4) DL_06(50)
	G02 X20 Y20 I0 J20 F2500	DL_17(10) DL_11(...)
	N60 G01 X100	HL_n05(1) DL_17(8) DL_06(100)
	N70 Y100	HL_n05(1) DL_17(2) DL_06(100)
	N80 M30	HL_n05(1) HL_End

Рассмотрим пример интерполятора, реализующего линейную, круговую и сплайновую интерполяции, а также их комбинации. Модуль интерполяции подключен к общей объектно-ориентированной магистрали системы ЧПУ. Интерфейс интерполятора обеспечивает прием параметров интерполяции и оперативных сигналов управления интерполятором, а также выдачу данных о состоянии модуля интерполяции (рис. 78).

Рис. 78. Интерфейс подключения интерполятора к общей магистрали

Схема интерполятора приведена на рис. 79 в виде некоторого набора блоков, собственной внутренней шины и администратора. Кадры управляющей программы поступают на вход транслятора в IPD-формате, преобразуются во внутренний формат интерполятора, обрабатываются в блоке опережающего просмотра кадров Look Ahead (с целью сглаживания скорости подачи) и запоминаются в кольцевом буфере. Транслятор формирует сообщения, в которых упакованы параметры интерполяции. Сообщения адресуются к определенным блокам интерполятора и могут быть главными и дополнительными. Главные сообщения содержат данные, необходимые адресуемым блокам, а дополнительные сообщения содержат данные о перемещениях вдоль координатных осей. Таким образом, главные сообщения инициализируют блоки, которые должны быть задействованы в интерполяторе при отработке кадра управляющей программы, а с помощью дополнительных сообщений инициализируются координатные оси, принимающие участие в интерполяции.

Внутренняя шина интерполятора является «шиной быстрых процессов» и связывает между собой все блоки. Она реализована на базе объектно-ориентированного подхода и соединена с основной объектно-ориентированной шиной системы ЧПУ с помощью администратора. Блоки, участвующие в отработке текущего кадра, назначаются с помощью специального кода. Этот код инициализируется в трансляторе и передается в администратор.

Код Mode блока интерполятора занимает 1 байт и состоит из двух частей:

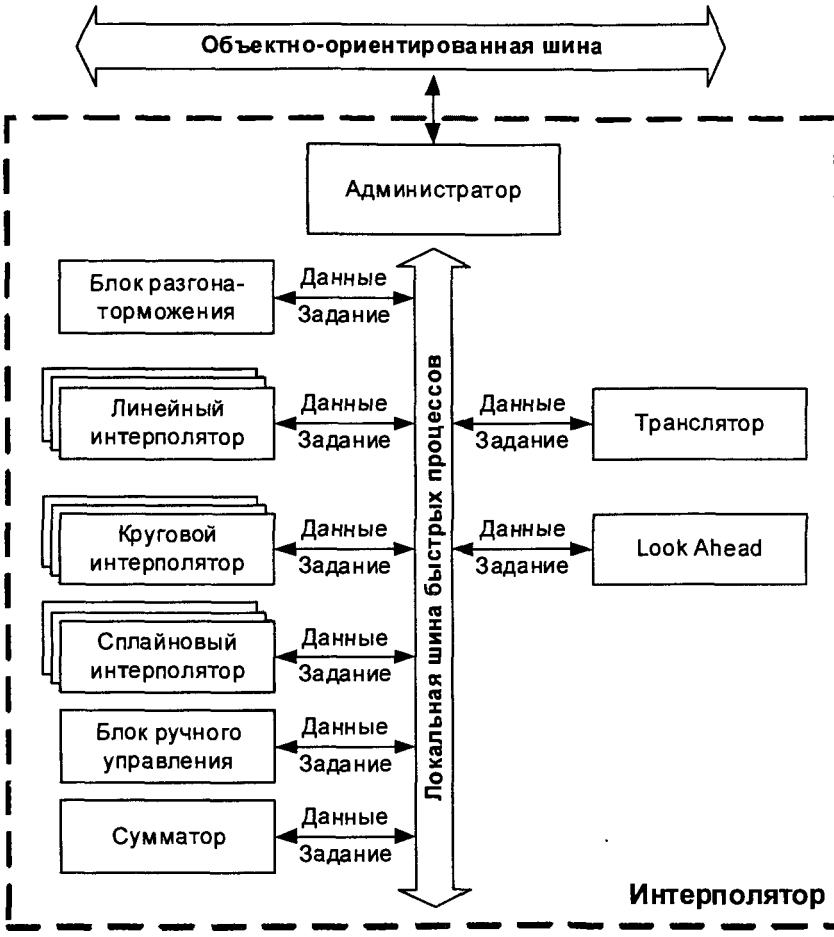


Рис. 79. Структурная схема интерполятора

Блоки интерполятора разделены по функциональному признаку на группы, как показано в табл. 7.

Номер блока и режим его работы определяются в трансляторе в зависимости от содержания кадра управляющей программы. В одном кадре после трансляции могут быть задействованы несколько блоков интерполяции из разных или одинаковых групп. Например, при воспроизведении на станке винтовой линии будут включены блоки линейного разгона-торможения (Mode = 1.0), круговой интерполяции (Mode = 3.1), линейной интерполяции (Mode = 3.0) и управления приводами подачи (Mode =7.0). При обработке деталей на пяти- шестикоординатных станках возможна одно-

Таблица 7. Кодирование блоков интерполятора

Группа	Mode	Назначение группы
0		Резервная группа
	0.0	Признак конца кадра
1		Блок управления скоростями и ускорениями
	1.0	Блок разгона-торможения
2		Резервная группа
3		Контурные интерполяторы
	3.0	Линейный интерполятор
	3.1	Круговой интерполятор
	3.2	Сплайновый интерполятор
4		Резервная группа
5		Резервная группа
6		Резервная группа
7		Блоки управления приводами подачи
	7.0	Сумматор

временная работа нескольких круговых и линейных или сплайновых интерполяторов.

Значения номеров блока и группы определяют приоритет модуля при ожидании в очереди к процессору. Блок с наименьшим номером имеет наивысший приоритет. Приоритеты блоков можно менять путем присвоения им других номеров.

Режимы работы блоков интерполятора задаются кодом CodeIn (табл. 8), который занимает один байт. Структура кода имеет следующий вид:

Таблица 8. Кодирование режимов интерполятора

Код	Режим интерполяции
Блок разгона-торможения (Mode = 1.0)	
0	Линейный закон разгона и торможения
1	Линейно-параболический закон разгона и торможения
2	Линейно-экспоненциальный закон разгона и торможения
Блок линейной интерполяции (Mode = 3.0)	
0	Быстрое позиционирование
1	Линейная интерполяция
Блок круговой интерполяции (Mode = 3.1)	
2	Круговая интерполяция по часовой стрелке
3	Круговая интерполяция против часовой стрелки

Подобная схема кодирования обеспечивает гибкость и открытость интерполятора. Возникает возможность построения администратора, который инвариантен к составу и количеству блоков интерполятора.

Назначение блока Look Ahead опережающего просмотра состоит в определении условного времени отработки кадра в циклах интерполятора для последующей коррекции контурной скорости и анализе в каждом кадре базовых параметров движения (вектора контурной скорости в начале и конце кадра, скорости по дополнительным координатным осям, пути в основной системе координат, радиуса кривизны траектории движения). В результате своей работы блок Look Ahead определяет скорость в конце кадра (конечную скорость) и новое значение контурной скорости подачи.

Общая схема работы интерполятора выглядит следующим образом. После предоставления кванта процессорного времени администратор (построенный по схеме микропрограммного автомата) посылает запрос транслятору на получение кодов блоков интерполятора, которые должны быть запущены. Получив коды, администратор запускает блоки в порядке их приоритетов, причем перед каждым запуском устанавливает одну из следующих команд: «ЗАГРУЗКА», «РАБОЧИЙ ТАКТ», «ЗАВЕРШАЮЩИЙ ТАКТ», «АВАРИЙНОЕ ТОРМОЖЕНИЕ», «ОСТАНОВ ИНТЕРПОЛЯТОРА», «СБРОС ИНТЕРПОЛЯТОРА», «ПУСК ИНТЕРПОЛЯТОРА».

Список команд можно изменять и расширять путем перепрограммирования администратора. Процесс загрузки нового кадра совпадает с завершающим тактом интерполяции предыдущего кадра. Завершающий такт интерполяции состоит в выходе в конечную точку траектории и не требует сколько-нибудь сложных вычислений. По команде «РАБОЧИЙ ТАКТ» каждый из интерполяторов обращается с запросом к блоку разгона-торможения и получает от него значение приращения пути, которое необходимо пройти вдоль контура в цикле интерполяции. Блок разгона-торможения запускается ранее интерполяторов, поскольку имеет более высокий приоритет.

Блок «СУММАТОР» формирует суммарные приращения пути из отдельных составляющих и выдает их на приводы подачи. Кроме того, сумматор накапливает абсолютные значения координат и хранит их в течение всего времени работы. По команде «УСТАНОВКА ФИКСИРОВАННОЙ ТОЧКИ» сумматор инициализирует абсолютные координаты. В этом же блоке работает и алгоритм коррекции погрешностей ходовых винтов.

Заключение

Геометрическая задача ЧПУ полагается традиционной, однако требования к ней за последнее время существенно изменились (усложнились). Эти изменения связаны с внедрением общей концепции открытых систем управления, а также с использованием объектно-ориентированной техно-

логии программирования, которая только одна в силах справиться с нарастающими сложностью и объемом математического обеспечения систем ЧПУ.

3.2. Реализация логической задачи управления

Изложен подход к реализации логической задачи числового программного управления мехатронными системами. В рамках жизненного цикла логической задачи рассмотрены фазы программирования, интерпретации программы и исполнения. На первой фазе представляется важным применение визуальных средств программирования, которые дают оператору инструмент для графической диалоговой разработки программы управления электроавтоматикой, причем эта программа интерпретируется в исполняемые C++ коды без компиляции в промежуточный язык. При этом изменяется (в сторону существенно большей эффективности) сама структура математического обеспечения системы логического управления.

Логическая задача, являясь по сути системой управления цикловой электроавтоматикой, реализуется двояко: программно в рамках системы ЧПУ или с помощью программируемого контроллера. Традиционный контроллер – это специализированный аппарат, дооснащенный терминалом в виде персонального компьютера. При этом возрастание мощности и уровня сервиса персонального компьютера позволяет объединить терминал, программатор и собственно контроллер в рамках единой компьютерной архитектуры с дополнительным модулем ввода-вывода сигналов электроавтоматики.

Существует прообраз, который называют системой PCC (Personal Computer-Controller – персональный программируемый контроллер). Прогнозируя развитие концепции PCC, можно постулировать такие ее особенности:

- использование однокомпьютерного варианта, с операционной системой Windows NT и расширением реального времени;
- увеличение числа функций интерфейса оператора за счет многорежимного управления и применения встроенных инструментальных систем программирования;
- поддержание в реальном времени динамических графических моделей (мнемограмм) управляемого объекта;
- построение терминальной части системы PCC по типу «виртуального прибора» [52, 53];
- применение визуального программирования электроавтоматики (например, по типу графического языка HighGraph фирмы Siemens, [54]) с генерацией C++ кодов исполняемого модуля; организация многопоточного управления (multi-thread).

3.2.1. Формализм описания циклов электроавтоматики

Для описания электроавтоматики воспользуемся формализмом иерархических графов, который удобен для графического описания циклов, в том числе и с помощью инструментальных средств визуального программирования [55]. Иерархический граф представляет собой «четверку» множеств:

- простых вершин-состояний, изображаемых кружками, причем состояния могут быть статическими или динамическими (выход из статического состояния инициируется извне, тогда как выход из динамического состояния происходит по завершению процесса);
- сложных вершин-состояний, изображаемых двойными (с двойным бордюром) кружками, причем такие состояния сами по себе являются вложенными графами;
- дуг, отражающих переходы между состояниями любого типа;
- узлов, «разрезающих» дуги, изображаемых темными кружками, причем узлы фиксируют условия смены состояний любого типа (если дуга исходит из статической вершины-состояния, то узел может одновременно принадлежать другому графу того же или другого уровня иерархии).

Вершинам-состояниям приписаны «этикетки» – имена в прямоугольных рамках. Имя статического состояния имеет структуру СТАТУС_<имя>. Имя динамического состояния имеет структуру ПРОЦ_<имя> или структуру ИНИЦ_<имя>. Имя узла служит признаком одного из следующих типов: команды, инициируемой с панели оператора, флага завершения вычислительного процесса, сигнала окончания управляемой операции.

Методика описания цикла электроавтоматики включает этапы: разработки первичного автомата, т.е. автомата верхнего уровня иерархии, являющегося по сути диспетчером режимов; разработки режима нерегулярных ситуаций (внутреннего режима), который сохраняет корректность состояния управляемого объекта при любых переключениях основных режимов, а также гарантирует неизменное состояние объекта, если цикл пассивен; выделения параллельно работающих автоматов, действующих в рамках цикла; разработки автоматов нижнего уровня иерархии.

Рассмотрим систему управления револьверной головкой токарного станка (рис. 80), которая поддерживает следующие режимы:

- автоматический, в рамках которого обеспечивается вызов инструмента любой грани револьверной головки (переход на нужную грань происходит по кратчайшему пути);
- ручного управления (например, поворот револьверной головки на очередную ее грань);
- нерегулярных ситуаций.

Двухпозиционный гидрораспределитель с электромагнитом ЭМ1 управляет гидроцилиндром ГЦ зажима-разжима револьверной головки, ниж-

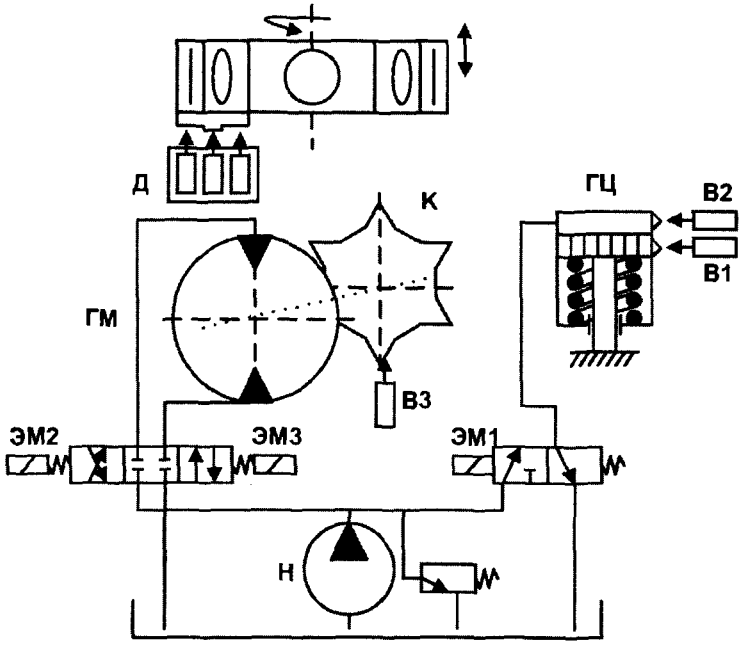


Рис. 80. Схема работы револьверной головки

нее и верхнее положения которой контролируются выключателями В1–В2. Поворот головки осуществляется гидромотором ГМ, управляемым гидрораспределителем с электромагнитами ЭМ2 и ЭМ3. Прохождение любой грани через возможное (по углу поворота) положение идентифицируется выключателем В3 и кулачком К, соосным с валом гидромотора. Код грани устанавливается многоразрядным датчиком Д.

Воспользуемся нашей методикой описания цикла. На рис. 81 приведен первичный граф (автомат) со всеми сложными состояниями, в том числе СТАТУС_НЕРЕГ_РЕЖ (режим нерегулярных ситуаций), СТАТУС_АВТОМ_РЕЖ (автоматический режим), СТАТУС_РУЧН_РЕЖ (режим ручного управления). Первичный автомат инициируется узлом-условием (клавишей) начального пуска НАЧ_ПУСК, а после этого управляет режимными переходами. Узлы-условия АВТОМАТ и РУЧН соответствуют активизации соответствующих режимных управляющих элементов панели оператора.

На рис. 82 раскрыта сложная вершина-состояние СТАТУС_НЕРЕГ_РЕЖ первичного автомата, которая сама по себе является графом режима нерегулярных ситуаций. Вход в граф осуществляется через один из узлов-условий: НАЧ_ПУСК, АВТОМАТ, РУЧН (см. рис. 81). Затем инициируется

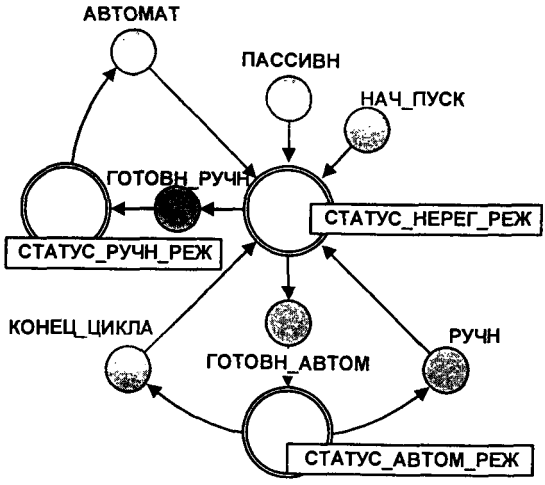


Рис. 81. Граф первичного автомата

сложная динамическая вершина-состояние ПРОЦ_ИДЕНТИФ, в рамках которой идентифицируется текущее состояние револьверной головки. Если головка вращается (узел-условие ВРАЩ), то вызывается алгоритм определения совпадения с корректным (по углу) положением грани (в динамическом состоянии ПРОЦ_ОПРЕДЕЛ_СОВПАД). При этом в корректном положении формируется команда торможения (узел-условие КОМ_ТОРМ). Если головка разжата или неподвижна (узел-условие РАЗЖАТ и СТОИТ), то вновь вызывается тот же алгоритм в вершине-состоянии ПРОЦ_ОПРЕД_СОВПАД, в зависимости от результата работы которого возбуждаются команды зажима (КОМ_ЗАЖ) или вращения (КОМ_ВРАЩ).

Наконец, если в качестве результата идентификации установлено зажатое состояние головки (узел-условие ЗАЖАТ), то система переходит в состояние подготовки к управлению (ПРОЦ_ГОТОВН). Подготовка завершается условием готовности ручного или автоматического режима.

Далее выделим параллельные процессы: зажим-разжим головки, вращение-останов головки (принцип выделения состоит в наличии собственного двигателя). Процессы отображаются независимыми графами, показанными на рис. 83 и 84. Они инициируются в сложном состоянии ПРОЦ_ИДЕНТИФ графа режима нерегулярных ситуаций.

Инициация зажима-разжима осуществляется в состоянии ИНИЦ_1 графа. Инициация устанавливает условие:

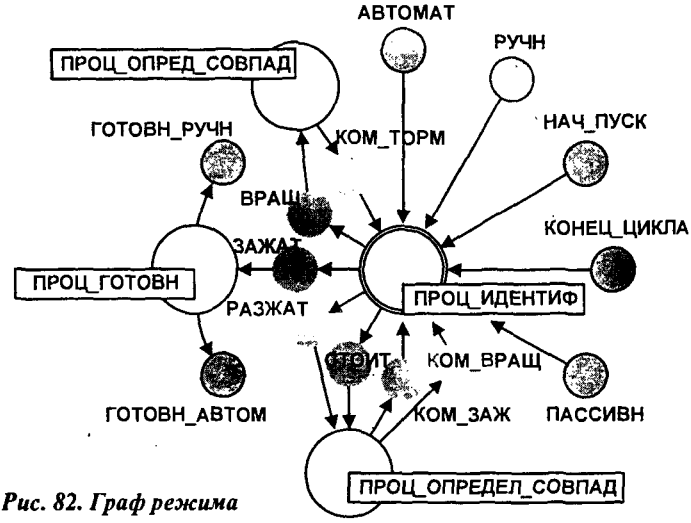


Рис. 82. Граф режима нерегулярных ситуаций

- корректный разжим (КОРР_РАЗЖ), что означает разжатое состояние головки в корректном положении ее граней по углу поворота (подобное условие инициирует процесс зажима в состоянии ПРОЦ_ЗАЖ);
- корректный зажим (КОРР_ЗАЖ), что означает зажатое состояние головки в корректном положении ее граней по углу поворота (подобное условие инициирует переход в статическое состояние зажима СТАТУС_ЗАЖ);
- некорректный зажим (НЕКОР_ЗАЖ), что означает зажатое состояние головки в неправильном положении ее граней по углу поворота – по причине сбоя (подобное условие инициирует процесс разжима головки в состоянии ПРОЦ_РАЗЖ);
- некорректный разжим (НЕКОР_РАЗЖ), что означает разжатое состояние головки в неправильном положении ее граней по углу поворота (подобное условие инициирует переход в статическое состояние разжима СТАТУС_РАЗЖ).

Инициация вращения-останова осуществляется в состоянии ИНИЦ_2 графа.

Инициация устанавливает условие:

- некорректное вращение (НЕКОР_ВРАЩ), что означает включенное состояние вращения гидромотора ГМ поворота головки при ее зажатом состоянии (по причине сбоя, подобное условие инициирует процесс торможения головки в состоянии ПРОЦ_ТОРМ);

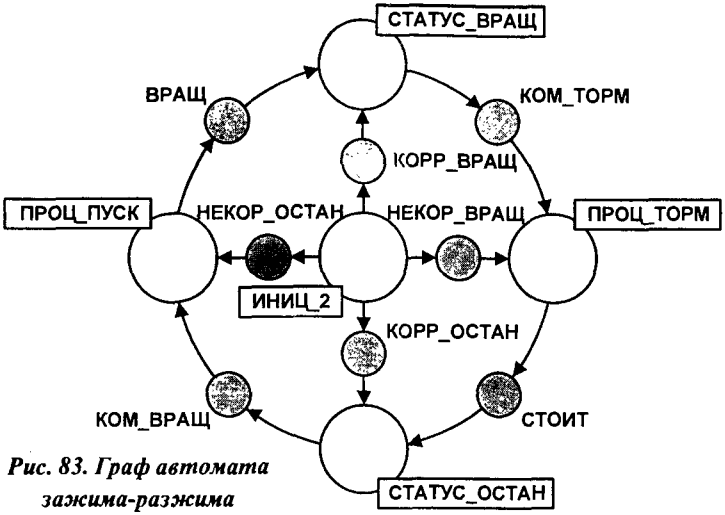


Рис. 83. Граф автомата зажима-разжима

• корректный останов (КОРР_ОСТАН), что означает остановку головки в корректном положении ее граней по углу поворота (подобное условие определяет переход в статическое состояние СТАТУС_ОСТАН);

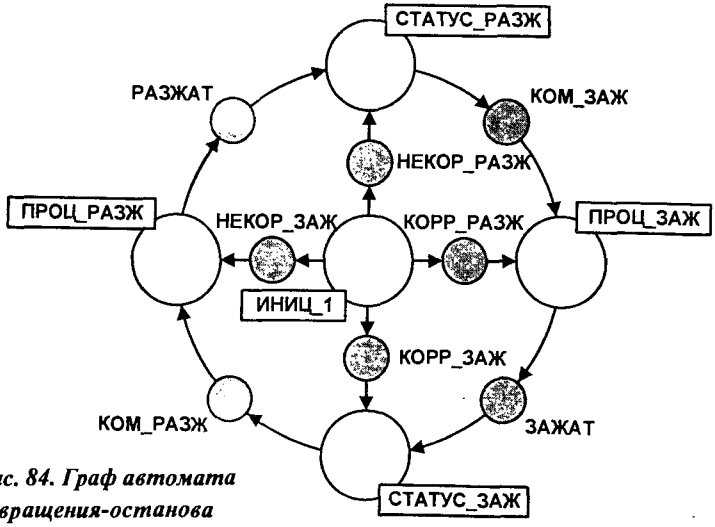


Рис. 84. Граф автомата вращения-останова

- некорректный останов (НЕКОР_ОСТАН), что означает остановку головки в неправильном положении ее граней по углу поворота (подобное условие инициирует процесс пуска вращения ПРОЦ_ПУСК);
- корректное вращение (КОРР_ВРАЩ), что означает нормальное вращение головки в ее разжатом состоянии (подобное условие определяет переход в статическое состояние СТАТУС_ВРАЩ).

Совместное рассмотрение двух графов на рис. 83 и 84 показывает, что система обеспечивает автоматический выход из любых некорректных ситуаций. Управляя условиями выхода из статических состояний, можно построить любой цикл автоматического или ручного управления.

3.2.2. Инструментальная поддержка визуального программирования циклов электроавтоматики

Однотипность скелета исполняемого кода циклов позволила разработать инструментальную систему визуального проектирования, генерирующую исполняемые C++ исходные файлы. Конкретный граф вводят с панели интерфейса программиста, которая предлагает набор графических примитивов: простую вершину-состояние, сложную вершину-состояние, дугу, узел дуги. Свойства примитивов (имена, типы вершин-состояний и др.) задают в диалоговом режиме на «странице свойств» (property page). Функции визуального проектирования обеспечивают: многоуровневое вложение графов с работой на каждом уровне в отдельном окне; выполнение групповых операций (выделение фрагмента графа, удаление, копирование, перемещение фрагментов в разных позициях и на разных уровнях); сохранение-загрузку проекта или фрагмента; импорт одного проекта в другой; документирование проекта и генерацию отчетов; генерацию исходного кода для последующей компиляции; верификацию графа на уровне проектирования, моделирование и отладку циклов. Применение инструмента визуального проектирования многократно повышает производительность разработчика, позволяет создавать сложные циклы электроавтоматики, реализация которых без инструментальной поддержки проблематична.

3.2.3. Генерация инструментальной системой C++ кодов исполняемых модулей циклов электроавтоматики

Диаграмма классов исполняемых модулей циклов электроавтоматики в нотации Booch [56] приведена на рис. 85. Диаграмма отражает состав и взаимоотношения классов. В соответствии с соглашением нотации каждый класс изображают в виде облака. Имя класса начинается с буквы «С» (например, CNcsState). Линии и стрелки показывают отношения между классами, а также структурами, типами и объединениями.

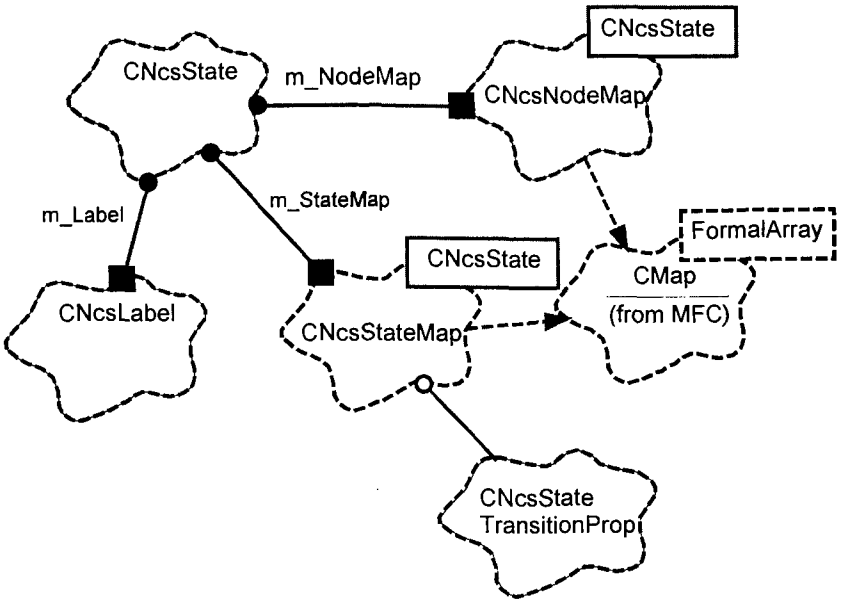


Рис. 85. Диаграмма классов в нотации Booch

Прототипом простого или сложного состояния служит класс CNcsState. Этому классу сопоставлена «этикетка« типа CLabel, которая сохраняется в m-поле m_Label – это показано отношением включения, в котором связь начинается закрашенным кружком и завершается закрашенным квадратом. Класс CNcsState хранит таблицу возможных переходов в m-поле m_NcsStateMap. Функциональные возможности таблицы отражены в реализации (instance) CNcsStateMap класса шаблона CMap библиотеки MFC, что обозначено штриховой стрелкой. Объект класса CNcsStateTransitionProp устанавливает структуру, тип и условия переходов. Класс CNcsState содержит указатель на таблицу вложенных состояний в m-поле m_NodeMap. Эта таблица инициализируется и заполняется только для сложных состояний. Таблица вложенных состояний отражена в реализации (instance) CNcsNodeMap класса шаблона CMap.

Заключение

Жизненный цикл логической задачи управления предполагает программирование, интерпретацию программы и ее исполнение. Современная тенденция состоит в упрощении первой фазы за счет визуального программирования, включая инструментальную поддержку, и в объектно-ориентированной реализации второй фазы.

3.3. Управление электроавтоматикой станков с ЧПУ по типу виртуальных контроллеров SoftPLC

На очередном витке эволюции программируемых контроллеров появилась и получила заслуженную популярность идея их программной реализации (SoftPLC). Наибольший эффект подобная идея дает в системах ЧПУ, где программное обеспечение виртуального контроллера SoftPLC работает в одной операционной среде с программным обеспечением ЧПУ. В этой связи возникает необходимость построения хорошо организованного и обозримого математического обеспечения виртуального контроллера на основе объектно-ориентированного подхода.

Сегодня появляется реальная возможность программной реализации управления электроавтоматикой станков в рамках общего программного обеспечения систем ЧПУ без привлечения дополнительной аппаратуры и системного программного обеспечения программируемых контроллеров, которые являются неотъемлемой частью практически любой современной системы ЧПУ. (Далее предполагаются системы ЧПУ, построенные на базе персональных компьютеров.) [3].

Подобные программные системы управления электроавтоматикой получили наименование виртуальных контроллеров SoftPLC. Указанный подход позволяет снизить стоимость системы управления при одновременном получении ряда преимуществ, в том числе упрощение общего программного обеспечения, уменьшение ошибок системного программирования, возможность отладки управляющих программ электроавтоматики в рамках самой системы ЧПУ, гибкость конфигурирования электроавтоматики, возможность использования различных коммерческих библиотек.

Далее предлагается объектно-ориентированный подход для построения виртуальных контроллеров электроавтоматики применительно к станкам с системами ЧПУ типа PCNC.

3.3.1. Объектно-ориентированный подход при организации математического обеспечения виртуальных контроллеров

В основе технологии создания программного обеспечения электроавтоматики лежат обычные для объектно-ориентированного программирования понятия класса и объекта. При этом класс описывает тип оборудования, а объект – конкретный экземпляр. Таким образом, при объявлении класса, согласно принципу инкапсуляции, создаются шаблоны структур данных и методы, которые будут работать с этими данными. В объекте класса по шаблону выстраиваются конкретные данные и приводится ссылка на обслуживающий их процесс.

При появлении нового типа оборудования, благодаря механизму наследования, разработчик не нуждается в том, чтобы заново разрабатывать новый класс – достаточно выбрать наиболее близкий и реализовать отличия в новом классе. Тем самым обеспечивается простота модификаций, сокращаются затраты времени на разработку, снижается общая стоимость разработки.

Наиболее важен тот факт, что объектный подход позволяет создавать хорошо структурированные сложные системы управления электроавтоматикой. Основные преимущества, приобретаемые при этом, состоят в следующем:

- повышается уровень унификации разработки; для повторного использования пригодны не только управляющие программы, но и проекты в целом, что служит хорошей основой для построения среды разработки. Снижаются затраты времени и средств на создание нового проекта;
- возникает возможность повторного использования собственных функциональных модулей и готовых модулей других разработчиков, что делает систему управления открытой. Уменьшается вероятность ошибок при разработке сложных систем, увеличивается уверенность в правильности принимаемых решений.

Все эти достоинства обеспечиваются благодаря лежащим в основе объектно-ориентированной технологии принципам наследования, инкапсуляции и полиморфизма.

3.3.2. Архитектура виртуального контроллера

В системе ЧПУ виртуальный контроллер работает в среде операционной системы Windows NT с расширением реального времени RTX фирмы VentureCom [4]. Проектирование контроллера предполагает последовательное рассмотрение его модели на трех уровнях абстракции: уровне модели потоков (структуры потоков), уровне функциональных модулей и уровне программной реализации (рис. 86).

Сначала рассмотрим структуру потоков. Основная задача контроллера состоит в одновременном выполнении нескольких команд и параллельной обработке внешних сигналов. Каждый процесс контроллера, который нуждается в выделении отдельного потока, выполняется в рамках основного процесса виртуального контроллера, запущенного под RTX. Процессорное время, выделяемое операционной системой основному процессу, должно быть распределено между потоками.

Далее воспользуемся идеей псевдомногопоточности на основе механизма выделения квантов (разделения времени). Процессорное время выделяется потокам отдельными квантами с помощью внутренних механизмов виртуального контроллера. В каждом кванте может выполняться толь-

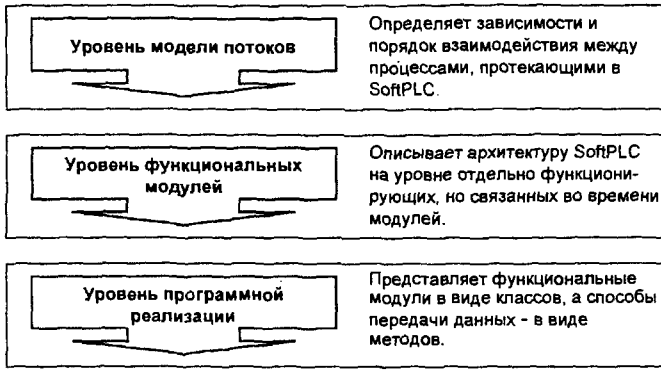


Рис. 86. Последовательная трансформация модели виртуального контроллера SoftPLC

ко один поток. Все потоки разделены на группы по приоритетам, причем управление группой осуществляется отдельным программным таймером. Программный таймер аналогичен системному, реализованному в операционной системе, но не генерирует прерывания, и его обработчик запускается планировщиком (в нашем случае – модулем синхронизации). Выделение нескольких групп потоков в виртуальном контроллере связано с тем, что различные его задачи требуют разного времени реакции на внешнее воздействие: чем меньше время реакции, тем выше приоритет потока, обслуживающего задачу.

Более высокий приоритет потока означает более высокую частоту выделения квантов времени. Различные частоты поддерживаются в системе несколькими таймерами, каждый из которых активизируется на своей частоте и выделяет кванты времени своим потокам. Модуль синхронизации осуществляет синхронную активизацию таймеров.

На временно́й диаграмме потоков, представленной на рис. 87, рассмотрен случай, когда виртуальный контроллер работает с тремя группами псевдопотоков.

Каждая группа получает кванты времени на выделенной частоте, что соответствует трем приоритетам. Таймер опорной частоты запускается на максимальной частоте сканирования входных регистров. Частоты программных таймеров формируются путем деления опорной частоты в обработчике событий таймера опорной частоты.

На временно́й диаграмме опорная частота равна 100 Гц, модуль синхронизации настроен на использование частот 100, 50 и 20 Гц. Для каждой

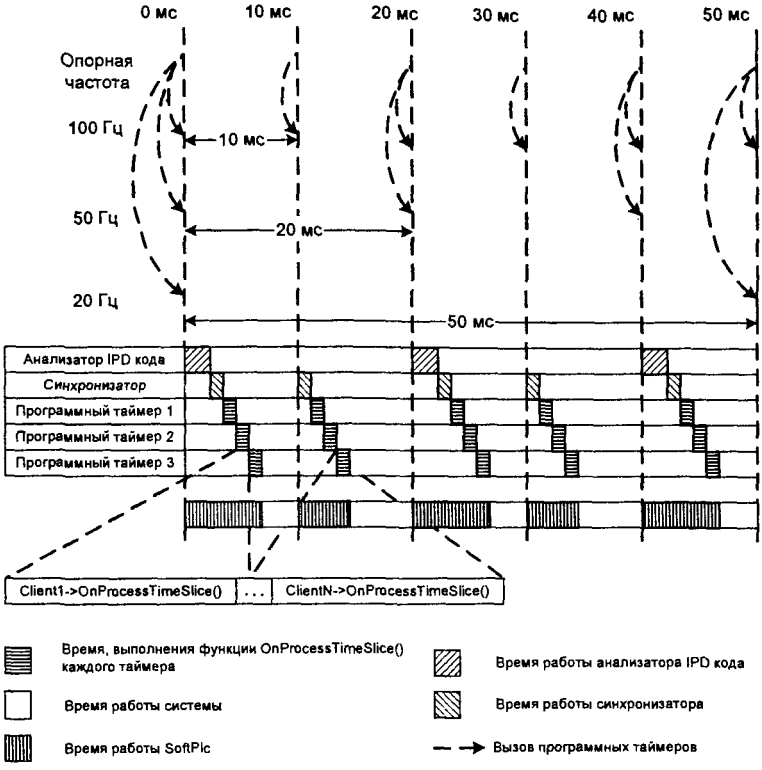


Рис. 87. Временна́я диаграмма потоков виртуального контроллера

частоты назначен программный таймер. Интервалы времени выделены для анализатора IPD-кода (InterPolator Data, данные на входе интерполятора), синхронизатора, каждого таймера в отдельности. Таким образом, в интервалах времени, кратных 10 мс, будут работать все три таймера.

С целью более равномерного распределения нагрузки в интервалах времени, задачи второго и третьего программных таймеров разделены, соответственно, на две и четыре подгруппы. Это позволяет запускать подгруппы поочередно в каждом интервале времени.

Модель контроллера на уровне функциональных модулей показана на рис. 88.

Виртуальный контроллер имеет пять составных частей (модулей):

- анализатор, читающий IPD-данные из входного буфера и преобразующий эти данные во внутренний формат виртуального контроллера с учетом входных и выходных регистров электроавтоматики;
- синхронизатор, поддерживающий механизм назначения квантов времени и генерирующий синхросигналы для всех процессов виртуального контроллера;
- исполняемые модули, служащие для отработки команд, поступающих в виртуальный контроллер, таких как опрос датчиков аварийного останова и конечных выключателей, включение/выключение подачи охлаждающей жидкости, зажим/разжим патрона, запуск/останов шпинделя, опрос датчиков температуры и т.д;
- регистр, используемый для обмена информацией между системой ЧПУ и виртуальным контроллером;
- шлюз, предназначенный для отображения информации, передаваемой по CAN-магистрали в регистр.

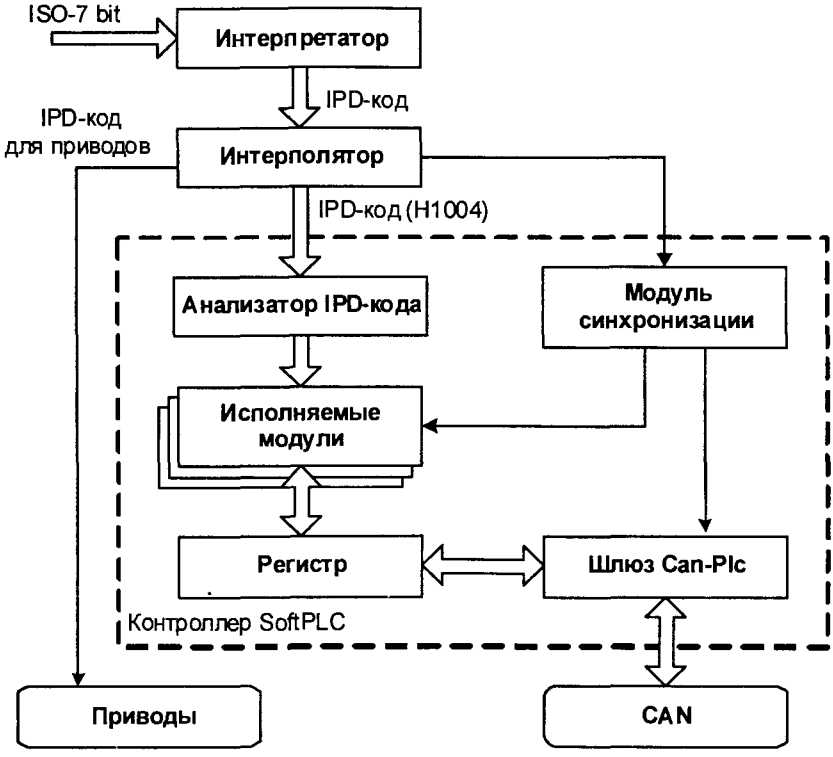


Рис. 88. Архитектура виртуального контроллера на уровне функциональных модулей

Взаимодействие модулей осуществляется следующим образом. В результате интерпретации управляющей программы ЧПУ формируется промежуточный IPD-код [21], представляющий собой универсальный бинарный код, не зависящий от используемой платформы. IPD-код содержит траекторную информацию об относительных перемещениях инструмента и детали, а также информацию о вспомогательных М-командах. Модуль интерполятора читает IPD-код, отделяя те данные, которые относятся к командам контроллера. Выделенная команда направляется соответствующему исполняемому модулю, внутри которого есть все необходимое для выполнения контроллером команды. Исполняемый модуль представлен в виртуальном контроллере в виде отдельной задачи.

Допускается параллельное выполнение нескольких М-команд. Механизм назначения квантов времени, генерирующий синхросигналы для всех процессов контроллера, обеспечивает синхронное выполнение команд. После отработки внутреннего алгоритма контроллер передает интерполятору информацию о своем состоянии.

Обмен данными между контроллером и системой ЧПУ осуществляется через разделяемую память, называемую в нашем проекте «регистром». В регистре выделены три блока: специальных маркеров SM (Special Marker), входных данных виртуального контроллера; выходных данных контроллера. Маркер SM используется для оповещения о нерегулярных ситуациях, возникающих в системе ЧПУ и контроллере. В каждом кванте времени блок SM анализируется на наличие в нем признаков сбоев (отказов аппаратуры и механизмов и др.) или признаков особо важных сигналов (при воздействии оператора на внешние органы управления). Например, аварийный останов принудительно прекращает работу всей системы. Информация о поступлении аварийного сигнала (при нажатии на кнопку аварийного останова) распространяется по всем объектам контроллера через определенную ячейку в блоке SM.

Блоки входных и выходных данных предназначены для организации двустороннего обмена с объектом управления: контроллер считывает информацию из блока входных данных и записывает ее в блок выходных данных. Передача информации между регистром и CAN-магистралью осуществляется посредством шлюза. Модель виртуального контроллера на уровне программной реализации рассмотрим в отдельном разделе.

3.3.3. Программная реализация виртуального контроллера

Виртуальный контроллер представляет собой некоторую систему с DLL-интерфейсом, работающую в отдельном RT-процессе реального времени (Real Time). Система имеет единственный экспортируемый класс CNcMParser, содержащий набор базовых функций управления и общедоступные экземпля-

ры входных и выходных данных. Остальные механизмы системы защищены от совместного доступа и не контролируются пользователем.

Объекты представляют собой экземпляры классов, описывающих электроавтоматику системы ЧПУ. В рамках модульной архитектуры виртуального контроллера каждый отдельный класс отвечает за свой объект управления на станке (рис. 89). Так, класс CNcSpindle отвечает за управление шпинделем, класс CNcClnt – за управление механизмом подачи смазочно-охлаждающей жидкости и т.д.

Благодаря модульной архитектуре виртуальный контроллер обладает высокой степенью гибкости, позволяющей использовать его на станках различных групп и типов. Конфигурация контроллера для заданного типа станка, в процессе ее инициализации, состоит в создании такого набора связанных объектов, который воспроизводит конфигурацию станка. Например, если в станке имеются два шпинделя, то будут созданы два объекта класса CNcSpindle.

При создании объектов осуществляется их взаимное связывание. На уровне языка C++ процесс связывания состоит в инициализации специальных указателей. Например, для привязывания патрона к шпинделю в

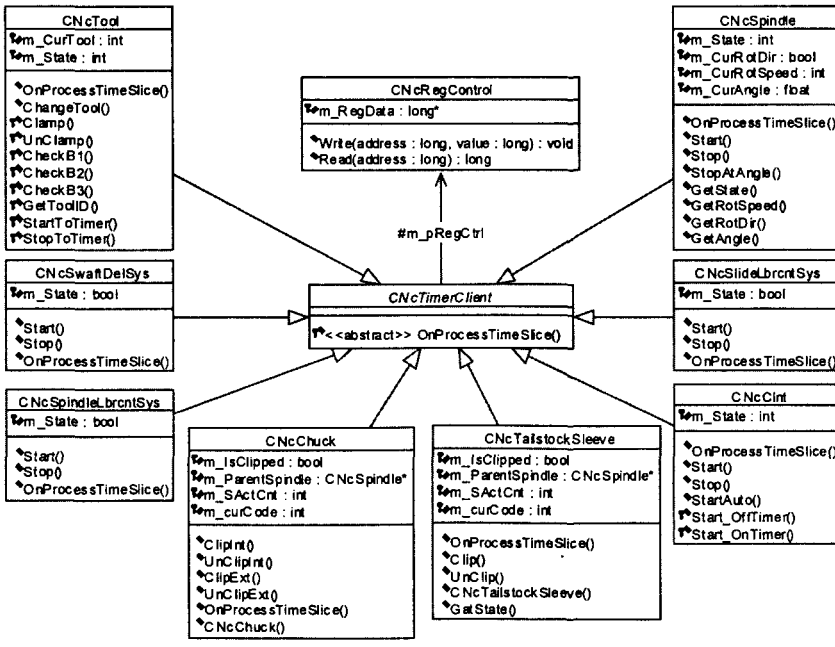


Рис. 89. Фрагмент диаграммы классов в нотации UML, реализующих исполняемые модули виртуального контроллера

классе CNcChuck инициализируется указатель m_ParentSpindle. Указатели инициализируются в конструкторах классов, т.е. информация о связях между объектами обновляется только при создании объектов.

После создания объектов и при необходимости настройки дополнительных параметров виртуальный контроллер переходит в состояние готовности выполнения команд. Рассмотрим программную реализацию функциональных модулей, входящих в состав виртуального контроллера (рис. 90).

Интерполятор вызывает анализатор IPD-кода с частотой 50 Гц, при этом анализатору передается некоторый объем IPD-кода с информацией о командах, которые контроллер должен выполнить. Анализатор описан отдельным классом CNcMParser.

Приемником IPD-кода, поступающего со стороны интерполятора по указателю на массив, служит метод ParserMCommand (long len, BYTE*

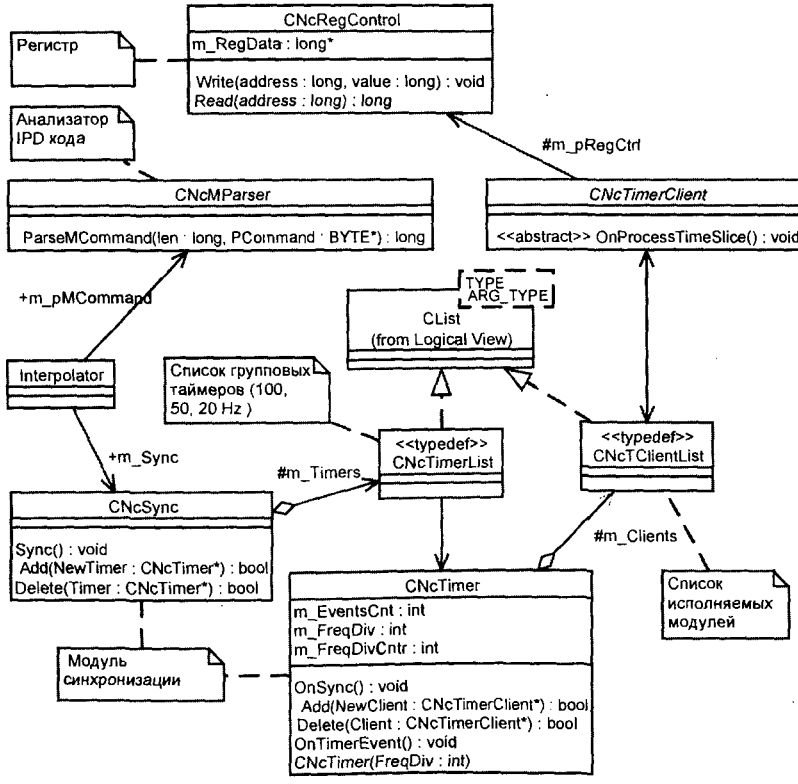


Рис. 90. Диаграмма классов, реализующих основные механизмы виртуального контроллера

PCommand). Объем кода указан в параметре «len». Метод ParserMCommand() интерпретирует IPD код, выделяя из него команды и их параметры. После выделения очередной команды разыскивается объект, для которого она предназначается (шпиндель, патрон, пиноль и т.д.), затем активизируется команда с заданными параметрами в соответствующем объекте. Активизация команды не означает мгновенного ее запуска: выполнение команды начнется в ближайшем кванте времени, выделенном объекту, и продолжится во всех последующих квантах вплоть до завершения реализации команды.

Модуль синхронизации предназначен для генерации в контроллере необходимого набора внутрисистемных частот, используемых для псевдопараллельного выполнения команд с помощью механизма квантов. Модуль синхронизации представлен классом CNcSync и построен на базе списка программных таймеров m_Timers. В класс CNcSync включены следующие методы:

- Sync() вызывается с каждым тактом опорной частоты и в свою очередь периодически вызывает методы OnSync каждого таймера, осуществляющего деление опорной частоты.
- Add(CNcTimer *NewTimer), предназначенный для добавления нового таймера в список. Каждый таймер содержит две переменные: время счета и счетчик времени. Эти переменные инициализируются при создании таймера, и таймер подключается к синхронизатору при помощи метода CNcSync::Add().

Синхросигнал опорной частоты передается с помощью метода CNcSync::Sync() модуля синхронизации (рис. 91). Метод CNcSync::Sync() просматривает список программных таймеров и для каждого элемента списка осуществляет вызов метода OnSync(). В методе CNcTimer::OnSync() увеличивается содержимое счетчика времени. Если значение счетчика време-

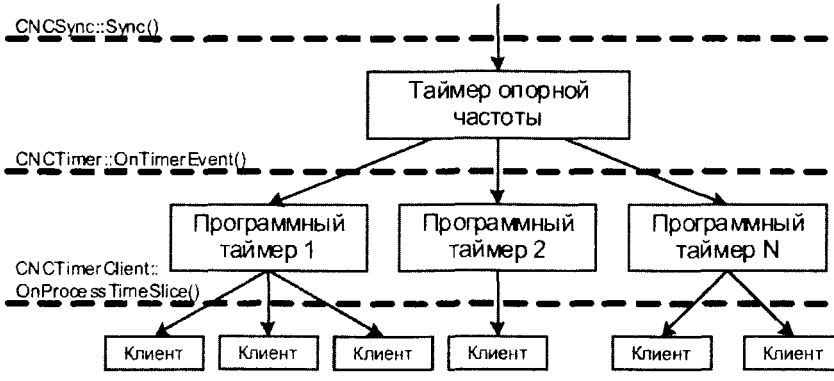


Рис. 91. Схема работы модуля синхронизации

ни становится равным времени счета, то счетчик времени обнуляется, после чего вызывается метод CNcTimer::OnTimerEvent().

Синхросигнал передается всем объектам, входящим в состав контроллера. Для этого в CNcTimer, на этапе инициализации контроллера создается список всех объектов m_Clients. Элементами списка служат указатели на объекты класса CNcTimerClient, порождающего все классы виртуального контроллера. Передача синхросигнала объекту (выделение кванта времени) осуществляется вызовом перегруженного метода OnProcessTimerSlice(): из списка m_Clients последовательно извлекаются все указатели на объекты. Каждый извлеченный указатель вызывает метод OnProcessTimeSlice(), в котором непосредственно отрабатываются команды.

Регистр, представляющий собой разделяемую память, поддерживает весь информационный обмен между аппаратными средствами и некоторыми объектами виртуального контроллера. Например, сигналы с датчиков отображаются в ячейках разделяемой памяти. Работа с регистром виртуального контроллера абстрагирована в специальном классе CNcRegControl. Методы CNcRegControl::Read() и CNcRegControl::Write() осуществляют чтение-запись содержимого регистра. Разделяемая память состоит из набора последовательно расположенных байтов, отображающих как текущее состояние физических объектов, подключенных к виртуальному контроллеру, так и вектор управления физическими объектами. Синхронизацию процессов чтения и записи осуществляет семафор.

Шлюз CAN-PLC предназначен для отображения информации, передаваемой по CAN-магистрали к регистру. Шлюз работает с интерфейсными функциями CAN-контроллера, считывает пакеты и направляет их в CAN-магистраль. Если некоторый флаг в регистре отображает состояние конечного переключателя, то шлюз будет принимать пакеты от конечного переключателя, анализировать их и обновлять этот флаг в регистре. Если некоторый байт регистра отвечает за управление устройством, подключенным к CAN-магистрали, то задачей шлюза будет слежение за изменениями этого байта и передача устройству значения байта.

3.3.4. CAN-интерфейс

CAN (Controller Area Network) представляет собой последовательную асинхронную шину, использующую витую пару в качестве среды передачи. Существуют две версии CAN-интерфейса: в версии A (BasicCAN) для идентификации сообщений выделены 11 битов, при этом система обсуживает до 2048 сообщений; в версии B (FullCAN) для идентификации сообщений выделены 29 битов, при этом система обсуживает до 536 млн сообщений.

CAN-интерфейс используют в распределенных системах для объединения интеллектуальных датчиков, интеллектуальных приводов и систем

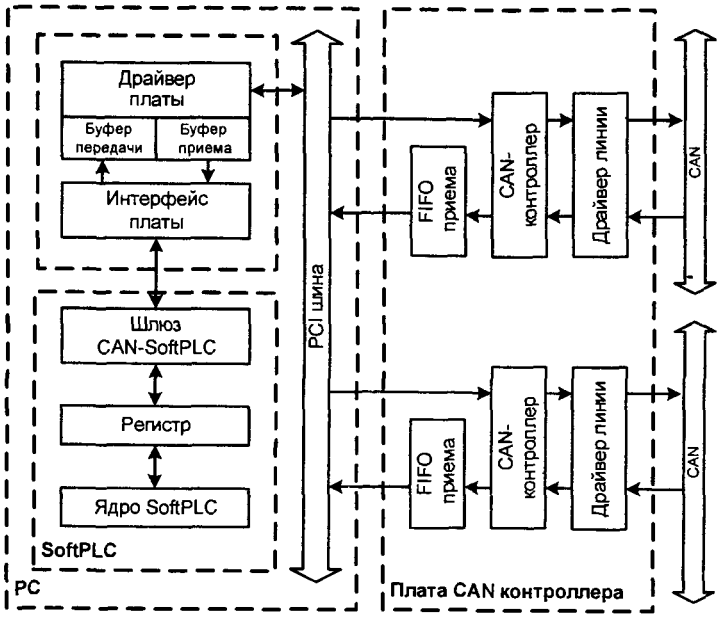


Рис. 92. Схема взаимодействия виртуального контроллера с CAN-магистралью

управления. Технологию CAN поддерживает и развивает некоммерческая международная группа CiA (CAN in Automation) [57]. В нашем случае CAN-интерфейс осуществляет передачу данных между разделяемой памятью и устройствами станка. В состав CAN-интерфейса входят контроллер, драйверы, сеть и удаленные устройства ввода-вывода (рис. 92).

Если один из абонентов сети передает информационный пакет, то прочитать его могут любые другие абоненты. Если CAN-контроллер обнаруживает признак начала передачи пакета, то считывает его во внутренний FIFO-буфер, из которого впоследствии пакет будет направлен в систему через драйвер контроллера. FIFO-буфер необходим, поскольку пакеты могут поступать быстрее, чем их считывает драйвер. Как только FIFO-буфер принимает очередной пакет, драйвер копирует его в буфер приема, откуда затем пакет может быть считан и проанализирован с помощью API-функций CanRcvMsg(). Если необходимо отправить пакет в сеть, то будет вызвана соответствующая API-функция CanSendMsg() CAN-интерфейса, ко-

торая сформирует пакет на основе полученных ею параметров и скопирует его в буфер передачи в памяти драйвера. Если драйвер обнаружит в буфере передачи новый пакет, то передаст его в контроллер и далее в сеть. Диаграмма последовательности считывания и записи данных в CAN со стороны исполнительного модуля в нотации UML [58] представлена на рис. 93.

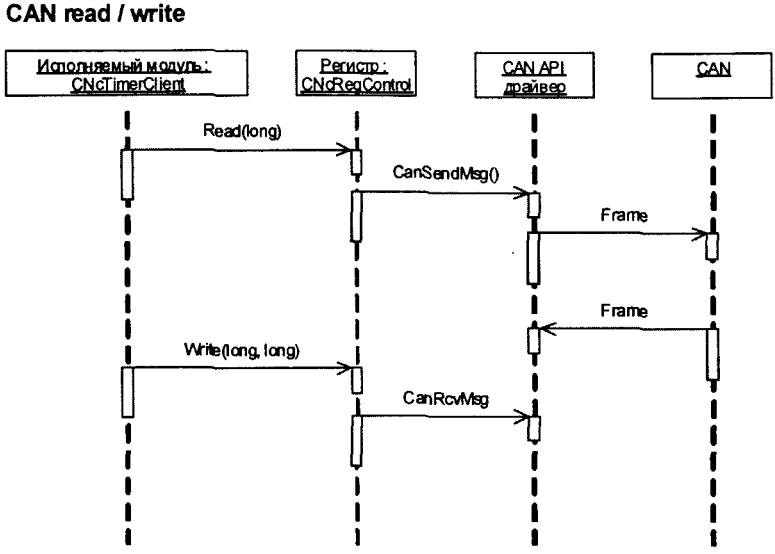


Рис. 93. Диаграмма последовательности считывания и записи информации в CAN-интерфейс

Скорость передачи данных в физическом канале CAN равна 1 Мбит/с. Фактическая скорость зависит от расстояния между абонентами, уровня помех и времени работы математического обеспечения канала. К передаваемым данным добавляется служебная информация, позволяющая идентифицировать пакет в узлах сети. Работа сети строится по принципу «каждый слышит каждого». При этом возможны два варианта информационного обмена (на примере датчика):

1. Датчик передает информацию о своем состоянии в сеть с некоторой частотой (например, 100 Гц). Контроллер обрабатывает поступающие данные по мере необходимости. Если они в этот момент не нужны, то игнорируются. Таким образом, ненужные данные все равно поступают в сеть, засоряя канал.

2. Датчик передает данные в сеть только по запросам, формируемым устройством. В нормальном режиме он следит за изменением своих параметров, но передает данные в сеть только при необходимости, т.е. после получения запроса. Запрос может быть однократным или групповым.

Наиболее удачный вариант состоит в том, чтобы найти оптимальную (различную) частоту потребности в данных от каждого датчика. В ряде случаев поток данных может достигать скорости, граничащей с пропускной способностью канала. Такие потоки лучше настраивать на групповую передачу по запросам.

Описанные методы относятся к низкоуровневым CAN-протоколам. Существуют протоколы и более высокого уровня, реализованные поверх нижнего, например CANopen. При их использовании возрастает сервис, но снижается эффективная пропускная способность.

CAN-контроллеры могут содержать несколько изолированных CAN-портов. В случае недостаточной пропускной способности одного порта можно использовать другие и тем самым повысить общую пропускную способность. Таким образом, создается несколько CAN-сетей, в каждой из которых могут находиться различные датчики. Недостаток заключается в том, что невозможен прямой обмен пакетами между сетями, т.е. абоненты, находящиеся в разных сетях, не могут обмениваться информацией без посредника. В качестве посредника (моста) выступает виртуальный контроллер, который транслирует пакеты из сети отправителя в сеть получателя.

Имеет смысл группировать датчики по сетям. Критериями могут служить интенсивность передачи информации датчиками, приоритет информации, принадлежность датчиков отдельным узлам оборудования.

Заключение

Идея построения виртуального контроллера SoftPLC на базе персонального компьютера чрезвычайно плодотворна и перспективна. Имеется весьма скудная информация о том, как строить ядро такого контроллера. Нами предложен объектный подход, благодаря которому удается достичь высокой степени обозримости системы, сократить затраты времени на разработку программного обеспечения и упростить процесс отладки. В рассматриваемой системе виртуальный контроллер SoftPLC имеет модульную архитектуру, в которой отдельный класс отвечает за свой объект управления на станке. Благодаря этому виртуальный контроллер обладает высокой степенью гибкости, позволяющей использовать его для станков различных групп и типов.

Основная задача виртуального контроллера SoftPLC, заключающаяся в одновременном выполнении нескольких управляющих команд и параллельной обработке внешних сигналов в режиме реального времени, была решена при помощи идеи псевдомногопоточности. Эта идея использует механизм разделения времени (выделения квантов), а также дополнительную возможность работы с приоритетами. Для информационного обмена с аппаратными средствами и между некоторыми объектами виртуального контроллера было предложено использовать разделяемую память.

3.4. Реализация терминальной задачи

Изложена формальная методика с применением инструментальных систем для реализации «скелета» терминальной задачи в Windows-интерфейсе. Рассмотрены конфигурируемые приложения в составе терминальной задачи для редактирования, отладки и моделирования управляющих программ в коде ISO-7bit и на языке высокого уровня.

Терминальная задача в составе математического обеспечения ЧПУ [59] имеет особое значение, поскольку предъявляет конечному пользователю функциональные возможности управления. Наполнение терминальной задачи определяет привлекательность и конкурентоспособность системы ЧПУ на рынке. Свойства открытой системы ЧПУ развиты настолько, насколько терминальная задача поддается конфигурации и расширению. Наиболее важными разделами терминальной задачи служат: интерпретатор диалога оператора в Windows-интерфейсе [55], редактор управляющих программ в коде ISO-7bit [61], редактор-отладчик управляющих программ на языке высокого уровня. Эти разделы и служат объектами дальнейшего рассмотрения.

3.4.1. Интерпретатор диалога оператора в Windows-интерфейсе

Современные системы управления используют архитектуру персонального компьютера и располагают широкими возможностями организации человеко-машинного интерфейса MMI (Man-Machine Interface) в операционных средах Windows NT или Windows 95/98. Терминальную задачу управления обычно сводят к проблеме построения MMI [21]; в этом случае задача выполняет функции клиента в клиент-серверной архитектуре математического обеспечения системы управления. Проектирование MMI-приложения предполагает создание скелета приложения, реализацию экранов, разработку интерпретатора диалога, организацию информационных сессий с другими модулями системы управления.

Этап разработки интерпретатора диалога наиболее сложен. В числе функций диалога можно обозначить: получение текущей информации о процессе управления; тестирование системы и объекта; редактирование и моделирование управляющей программы; ручной ввод и управление отработкой данных; ввод программы и автоматическое управление; управление наладочными операциями. Диалог устанавливает допустимые переходы между состояниями MMI-приложения, в рамках которых и воспроизводятся необходимые функции. Оператор системы управления задает переходы между состояниями с помощью аппаратной и функциональной кла-

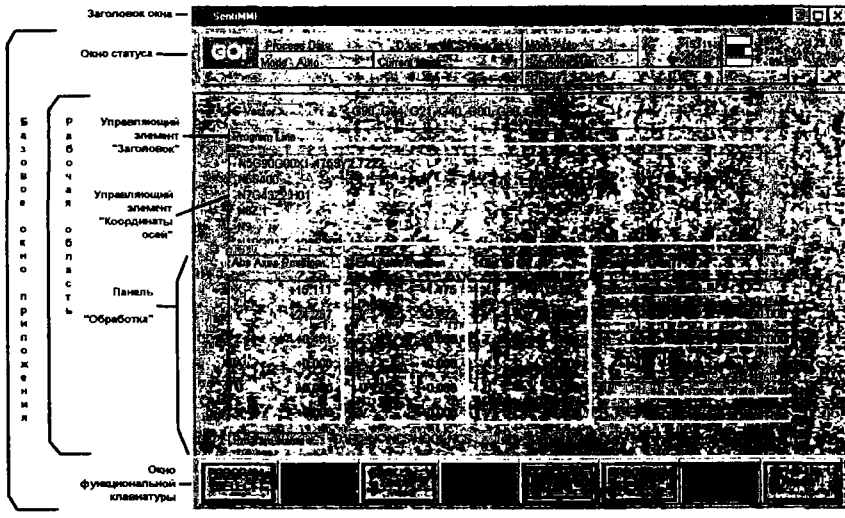


Рис. 94. Пример экрана MMI-приложения

виатуры, причем роль последней преобладает. Заметим, что использование мыши, как правило, недопустимо. На рис. 94 показан пример экрана, в котором размещается базовое окно (frame window) MMI-приложения. Базовое окно состоит из трех компонентов: окна статуса (StatusBar), рабочей области, области функциональной клавиатуры (ToolBar).

Рабочая область базового окна покрыта окнами-панелями с рамками и заголовками; каждая панель предоставляет функционально-однородную информацию. Панель в свою очередь содержит управляющие элементы (control elements), которые предназначены для динамического отображения данных, поступающих от сервера или вводимых оператором с помощью клавиатуры. Управляющие элементы могут быть выбраны на этапе проектирования экрана из стандартной библиотеки (галереи управляющих элементов) или подлежат проектированию в соответствии с требованиями заказчика.

Окно статуса точно так же состоит из панелей с управляющими элементами: специфическими (пиктограммами режимов и подрежимов, индикаторами готовности, индикаторами используемых системных ресурсов, индикаторами даты и времени) или общего характера (такими же, как и в рабочей области, -- для слежения за информацией закрытой при смене картинки рабочей области).

Область функциональной клавиатуры представлена кнопками, каждая из которых является управляющим элементом, имеющим динамическое имя. Типы и состояния кнопок показаны на рис. 95. Традиционный тип

кнопки – «Клавиша», нажатие которой переводит систему в новый режим (подрежим) или открывает диалоговое окно. Кнопка «Ввод» работает так же, как и соответствующая клавиша на панели оператора. Кнопка типа «Функция» инициирует выполнение некоторой функции без предварительных запросов и подтверждений. Кнопка «Ввод функции» объединяет возможности двух предыдущих кнопок. Кнопка «Селектор» выбирает одну из двух функций (например, метрическую или дюймовую размерность). «Мультиселектор» представляет собой группу, в которой активизация одной кнопки деактивирует все остальные. Кнопка «Триггер» последовательно активирует и деактивирует выбранную функцию. Все кнопки могут находиться в следующих состояниях: готовности, когда они доступны оператору («ненажатое» состояние); работы под воздействием оператора («нажатое» состояние); блокировки, когда они отсутствуют в области функциональной клавиатуры или когда воздействие на них оператора игнорируется («заблокированное» состояние).

Структура диалога в гибкой системе управления с открытой архитектурой определяется заказчиком системы ЧПУ, для которого привычным языком внешнего описания диалога служит «дерево» режимов и подрежимов. Ветвям «дерева» приписаны имена кнопок функциональной клавиатуры. На рис. 96 представлен фрагмент «дерева» для режима автоматического управления в одной из систем ЧПУ. Недостатки подобного представления состоят в отсутствии возвратов к началу диалога или его ранним

Рис. 95. Типы и состояния кнопок

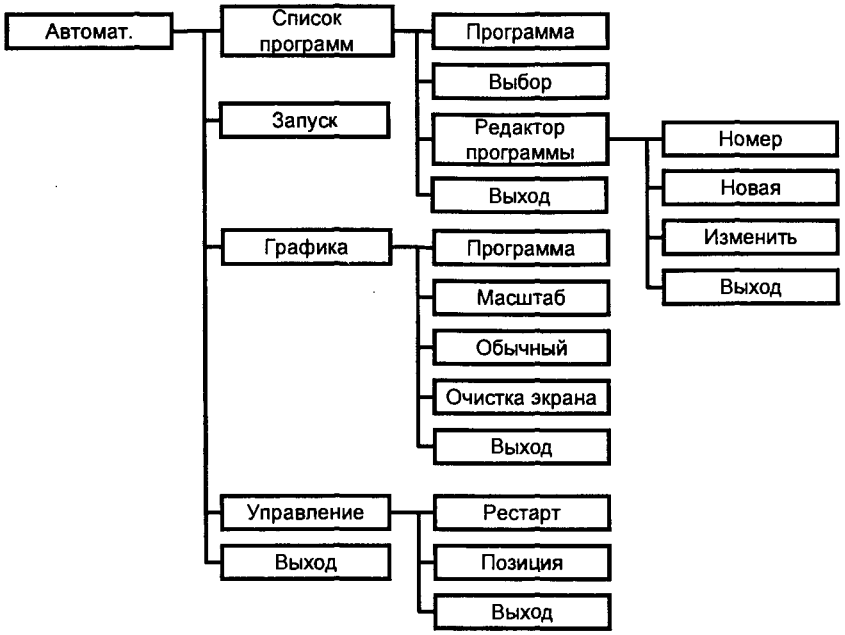


Рис. 96. Фрагмент «дерева» для режима автоматического управления

стадиям, в плохой обозримости «дерева», а также в том, что «дерево» не поясняет структуры функциональной клавиатуры.

В этой связи предлагаем описывать диалог, как это показано на рис. 97, для того же его фрагмента. Формальной моделью служит иерархический граф состояний, вершины которого отражают устойчивые состояния MMI после нажатия оператором той или иной клавиши панели оператора, а дуги нагружены именами функциональной клавиатуры или других клавишей.

Сложное состояние (двойной кружок) само по себе является графом, который раскрывается так, как показано в прямоугольном контуре на рис. 98. Иерархические графы удобны для описания многорежимных многоуровневых диалогов и позволяют проектировать диалог «шаг за шагом» – от укрупненного выбора режимов к детальному определению поддерживаемых функций. При этом возможно описывать процесс управления не только с использованием функциональной клавиатуры, но и с помощью меню, дерева навигации и т.д.

В графе состояний могут быть определены две группы переходов:

- транзитивные, связанные со сменой состояний;
- нетранзитивные, отображаемые дугой, возвращающейся в прежнее состояние.

Переход любого типа инициируется нажатием клавиши, имя которой приписано переходу (дуге). Поскольку каждая клавиша генерирует свой собственный код (сканкод), этот код также сопоставляется переходу (дуге). По своей сути переход есть запрос на услугу, которая содержательно представляет собой смену экранов, обновление имен клавиатуры экрана, обращение к серверу и др. Услуги реализуются объектами, которые работают в состояниях MMI (имена классов объектов обозначают состояния графа, рис. 97). Таким образом, ход диалога представляет собой последовательность следующих событий:

- нажатие клавиши оператором и генерация сканкода;
- обращение к объекту (для транзитивных переходов – в новом состоянии; для нетранзитивных переходов – в старом состоянии) с запросом на услугу (или услуги);
- реализация услуг объектом;

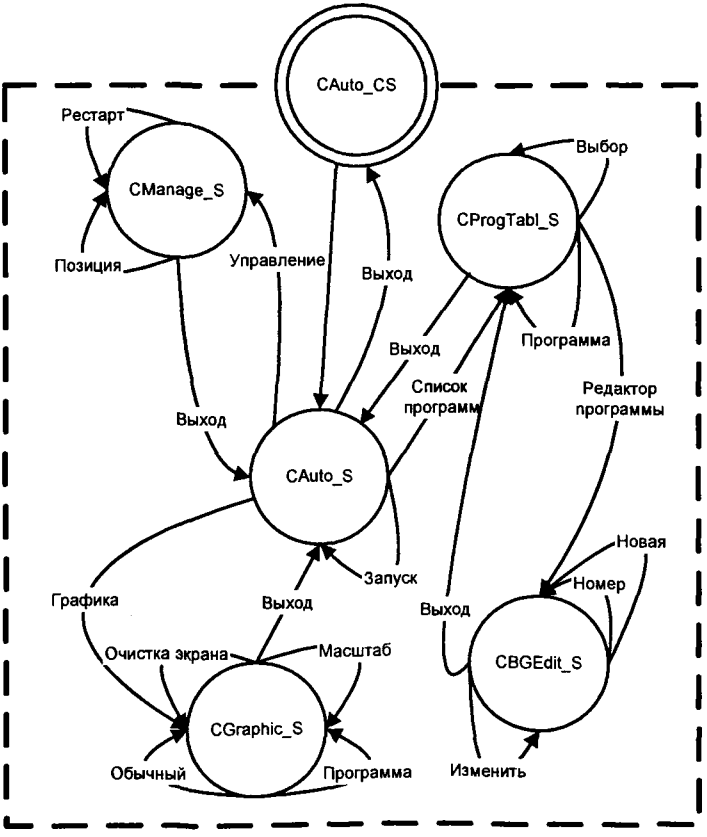


Рис. 97. Фрагмент графа режима автоматического управления

- ожидание очередных действий оператора.

Эту цепочку называют последовательностью актов [62], в числе которых есть входные (нажатия клавишей) и выходные (все остальные). Выходные акты порождаются в результате интерпретации входных.

Следовательно, интерпретатор диалога есть механизм последовательного вызова услуг соответственно действиям оператора, выражающимся в нажатии клавишей. Помимо вызова услуг интерпретатор осуществляет синтаксический контроль действий оператора. Для генерации C++ кодов интерпретатора разработана инструментальная система State Machine Builder. На рис. 98 показано рабочее окно инструментальной системы.

В рамках инструментальной системы визуального проектирования задание вводят непосредственно в виде иерархического графа. В диалоговом режиме устанавливают имена состояний и свойства переходов. Глубина вложения назначается разработчиком по его усмотрению, что позволя-

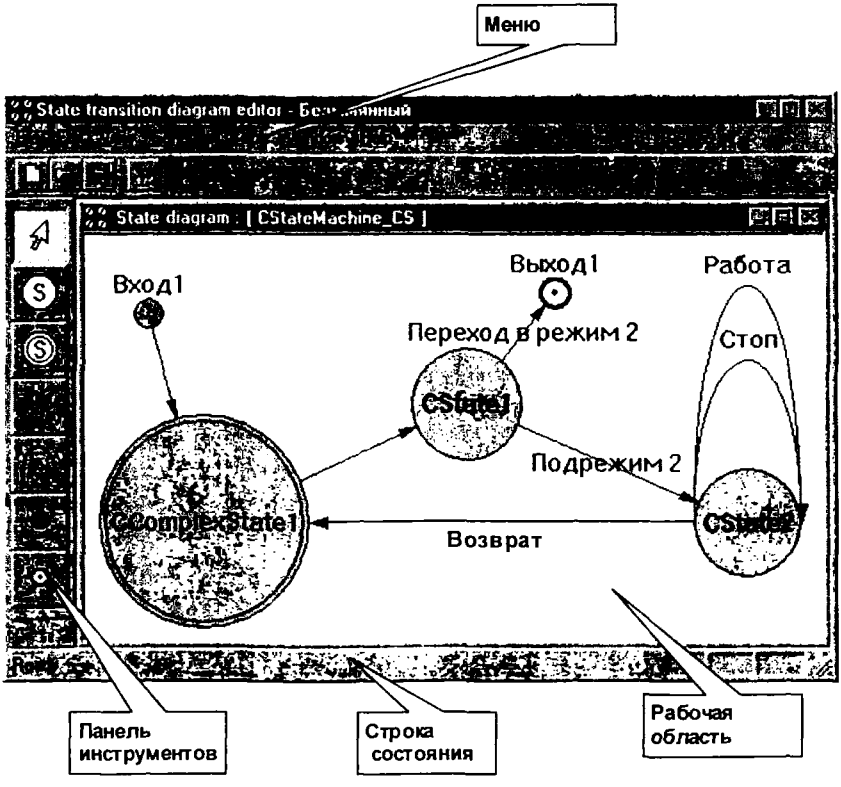


Рис. 98. Рабочее окно инструментальной системы генерации C++ кода интерпретатора диалога

ет концентрировать внимание на текущих фрагментах диалога. Разработчик должен также обозначить имена файлов, в которых будут сгенерированы классы состояния и т.д. Применение системы визуального проектирования многократно повышает производительность разработчика, позволяет создавать, а впоследствии и развивать сложные интерпретаторы, реализация которых без инструментальной поддержки весьма проблематична.

3.4.2. Специфика построения редактора управляющих программ в коде ISO-7bit (в составе терминальной задачи)

К редактору управляющих программ предъявляют как стандартные требования, характерные для текстового редактора, так и ряд специфических требований. К стандартным требованиям относятся:

- ввод и редактирование текста, скроллинг и перелистывание страниц; операции перехода, контекстного поиска и замены;
- блоковые операции маркировки, удаления, копирования, перемещения, загрузки и добавления блоков.

К специфическим требованиям относятся:

- перенумерация после изъятия-включения кадров;
- изменение масштаба и размерности;
- вывод активных G-функций (G-вектора) на основе предыстории кадра;
- синтаксический и семантический контроль;
- диалоговый (графический) ввод кадра и параметров стандартных циклов (файлы графической помощи находятся в составе конфигурационного файла);
- создание управляющих программ (УП) в режиме обучения [61].

Средства отладки программ включают:

- пространственное графическое моделирование траектории инструмента с различением (по цвету, типу и толщине линий) быстрых и рабочих перемещений;
- активное использование точек останова (break points), используемых, в том числе, для выделения фрагментов графического изображения;
- масштабирование графического изображения (zooming);
- поддержку различных режимов изображения (пошаговый, автоматический, между точками останова, со skip-пропуском);
- моделирование оставшейся части программы по отношению к текущей позиции станка.

Подобные возможности требуют включения в состав редактора некоторого ядра и дополнительных подсистем: интерпретатора управляющих программ (для любых версий кода ISO-7bit) и имитатора интерполятора для рисования траекторий.

Говоря о редакторе, необходимо затронуть проблему представления управляющих программ в коде ISO-7bit. Стандарт этого кода, принятый в 1970-х годах, практически не перетерпел изменений и тормозит использование сложных алгоритмов интерполяции (таких, как сплайновая интерполяция в реальном времени), управление лазерной и электроэррозионной обработкой и др. По этой причине производители систем ЧПУ используют собственные версии кода ISO-7bit в соответствии с потребностями своего круга пользователей. Многие версии не имеют четкой структуры, а их синтаксис базируется скорее на исключениях, чем правилах, поскольку версии создавались без общей концепции и расширялись стихийно. Тем не менее код ISO-7bit остается действующим стандартом и ни одна система ЧПУ не может его игнорировать. Заметим также, что все CAD-CAM системы генерируют выходной файл в формате ISO-7bit. В этой связи существует потребность в редакторе, конфигурируемом под конкретную версию кода ISO-7bit.

Конфигуратор формализует код ISO7-bit путем выделения в нем нескольких уровней абстракции. На первом уровне определяется система команд (G-функций) и параметры каждой команды. Следующий уровень разбивает систему команд на группы по функциональному назначению G-функций и формирует G-вектор активных команд. Последний уровень абстракции назначает списки разделителей, комментариев, имен осей и адресов, имен G-функций. Подобным способом удается формализовать любую версию кода ISO7-bit и соответствующим способом сконфигурировать редактор.

Основной принцип реализации редактора состоит в его модульности (рис. 99), которая позволяет применить стандартные Windows-решения (механизм «документ-представление», ActiveX-элементы в интерфейсе оператора, концентрацию ресурсов в ресурсных библиотеках для локализации на разных языках), использовать объектно-ориентированный подход, при котором каждому компоненту структуры редактора сопоставлен свой собственный набор объектов.

Первый модуль представляет собой клиентскую часть, определяющую, будет ли редактор независимым приложением или «встроенным» в режим системы ЧПУ. Второй, серверный модуль, выполненный в виде DLL-библиотеки, предлагает клиентской части полный набор сервисных функций и «скрывает» от клиентской части наличие других модулей. Виртуальная ISO-машина, построенная в виде DLL-библиотеки, выполняет такие операции над кадрами, как синтаксический и семантический контроль, конвертацию и интерпретацию кадра, вычисление G-вектора и т.д. Компоненты графического моделирования траектории инструмента также выделены в набор модулей и реализованы в виде ActiveX-элементов.

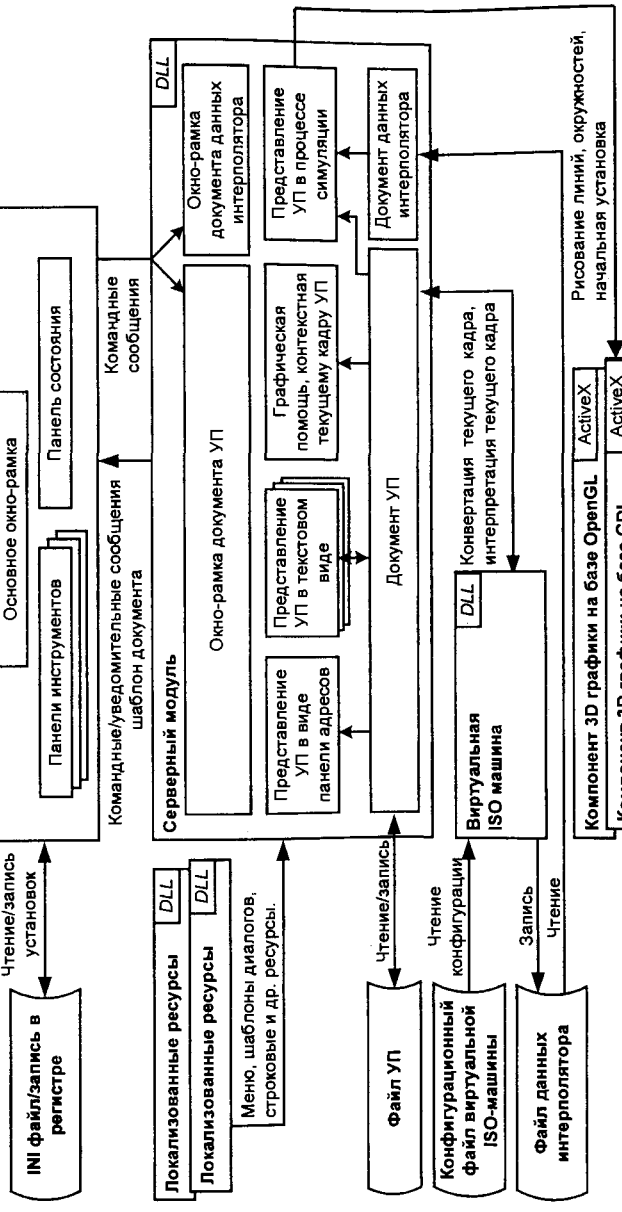


Рис. 99. Модульная структура NC-редактора

Виртуальная ISO-машина считывает конфигурационный файл (*.cfg) и настраивается на текущую версию кода ISO-7bit. Документ управляющей программы серверного модуля осуществляет работу с файлами управляющей программы. Файл загружается в документ и отображается (представляется) по-разному в зависимости от текущего режима редактора. Так, одни и те же данные могут быть представлены в виде панели адресов, в текстовом или графическом формате или в виде графической модели траекто-

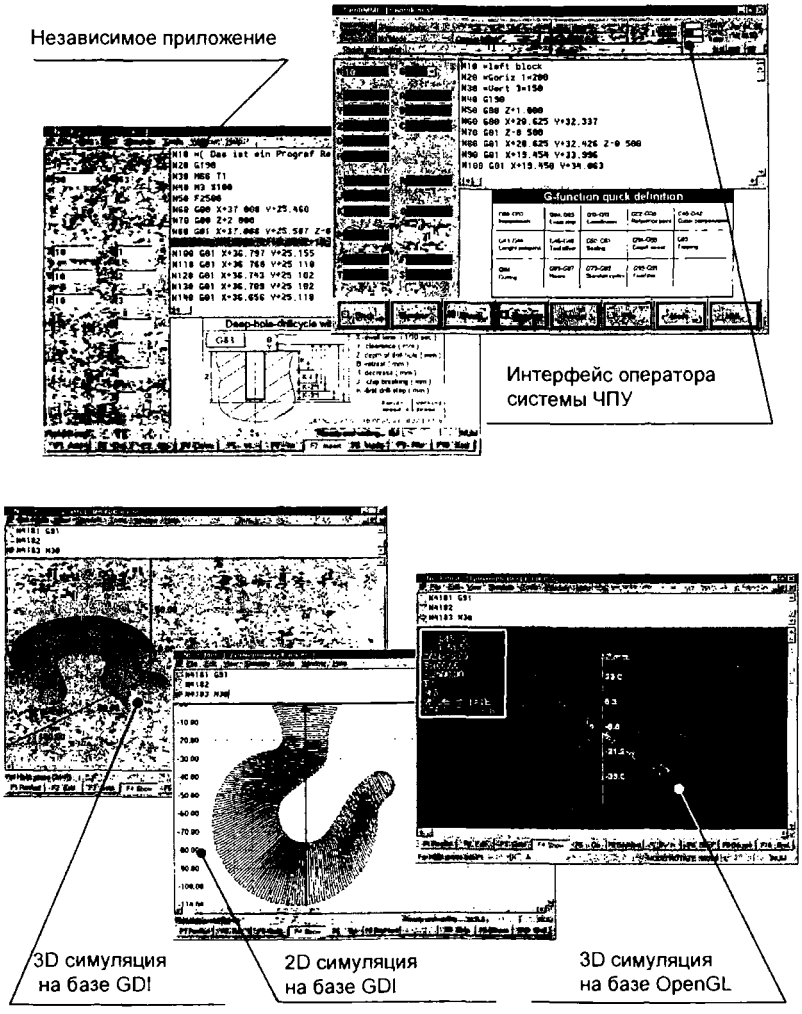


Рис. 100. Версии ActiveX-управляющих элементов для системы графического моделирования траектории инструмента

рии. Для интерпретации и конвертации кадра документ обращается к виртуальной ISO-машине. После интерпретации кадров виртуальная ISO-машина генерирует входной код интерполятора, IPD-код (InterPolator Data) и сохраняет его в файле. IPD-файлы считываются IPD-документом, который отображается в представление графического моделирования.

Клиентская часть генерирует функциональную клавиатуру, которая позволяет управлять редактором, а также окно статуса, где в процессе работы отображается контекстно-зависимая информация.

Редактор управляющих программ имеет архитектуру, открытую для конечных пользователей, разработчиков самого редактора, внешних приложений. Для конечных пользователей это прежде всего означает возможность конфигурации на различные версии языка ISO-7bit с помощью конфигурационного файла, имеющего текстовый формат. Далее существует возможность конфигурировать интерфейс пользователя, включая систему контекстных подсказок и систему помощи, используя текстовый файл инициализации и соответствующие динамические библиотеки ресурсов.

Разработчикам редактора предлагается архитектура, открытая для интеграции и компоновки. Так, редактор может быть интегрирован в существующий интерфейс системы ЧПУ или работать в качестве независимого приложения в технологическом отделе подготовки управляющих программ. Редактор предусматривает различные компоновки с ActiveX-элементами для графического моделирования траектории инструмента. На рис. 100 показаны 2D, 3D и OpenGL версии ActiveX-элементов, которые позволяют выбирать отображаемые оси, настраивать координатную систему и область просмотра, выбирать оптимальные значения, фильтровать набор переменных для просмотра их значений (текущих координат, имен инструментов, активных M- и G-функций и т.д.).

Архитектура, открытая для внешних приложений, поддерживается интерфейсом OLE IdataObject, с помощью которого осуществляется передача данных через «clipboard» – стандартный Windows-механизм для обмена данными между приложениями.

3.4.3. Редактор-отладчик управляющих программ на языке высокого уровня (в составе терминальной задачи)

В числе языков высокого уровня управляющих программ можно упомянуть AnlogC (фирма Andron, Германия), CPL (фирма Bosch, Германия) и множество других. Независимо от версии структуры всех языков однотипны: имеется основная программа и некоторый набор вызываемых подпрограмм. В теле программы представлен список переменных, которые по ходу реализации программы меняют значения. Процесс выполнения со-

провождается информационными сообщениями, предупреждениями, сообщениями об ошибках.

На рис. 101 показан экран редактора-отладчика. В окне программы отображается ее текст; здесь же можно расставить точки останова (breakpoints), осуществить пошаговый или автоматический запуск программы. Переменные (или массивы переменных) представлены в отдельном окне в виде «дерева», которое позволяет выбрать те из них, за значениями которых необходимо следить. Текущие значения выбранных переменных (или массивов) демонстрируются в другом окне, причем значения эти можно редактировать.

Иерархия вызываемых подпрограмм показана в виде «дерева» в отдельном окне. В окне стека представлены подпрограммы, вызванные к текущему моменту. Окно OutputWin используется для вывода информации, тип которой определяется неким набором пиктограмм. Пиктограммы предназначены также для отображения свойств файлов вызываемых подпрограмм.

Указанные окна относятся к числу базовых и постоянно присутствуют на экране. Окна со вспомогательной информацией (например, списком точек останова) реализованы как всплывающие. Основные и вспомогательные окна редактора-отладчика образуют ActiveX-управляющий элемент.

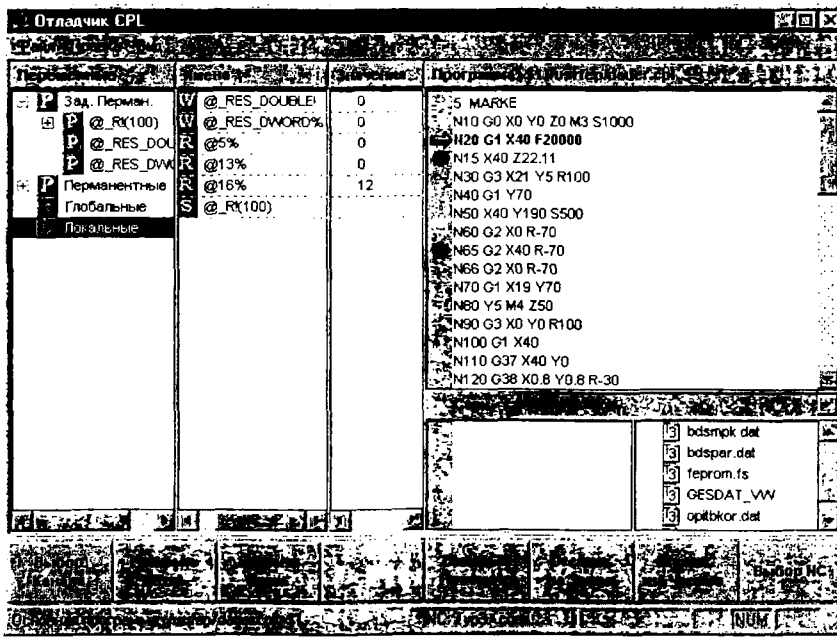


Рис. 101. Окно редактора-отладчика

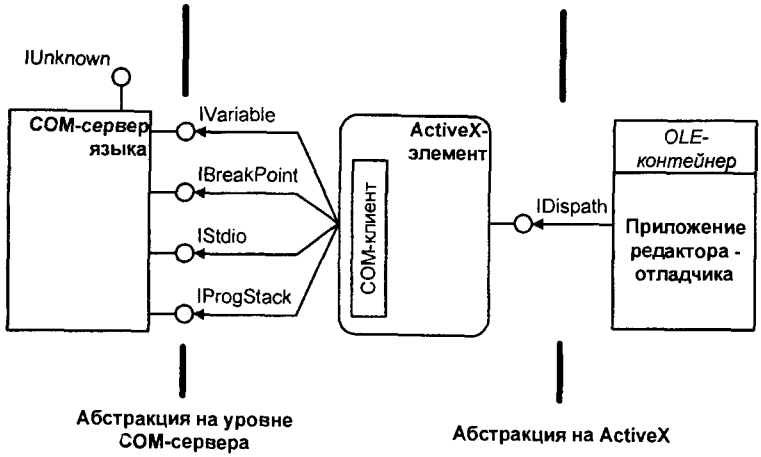


Рис. 102. Архитектура редактора-отладчика

Архитектура редактора-отладчика включает COM-сервер, ActiveX-управляющий элемент, приложение (рис. 102). В этой архитектуре выделены два абстрактных уровня. На первом уровне поддерживается работа с различными языками управляющих программ, причем для каждого языка необходимо разработать его собственный COM-сервер. Любой COM-сервер, однако, должен располагать неизменным набором интерфейсов для работы с переменными, точками останова, буферизованными файлами и сообщениями. В этом случае ActiveX-управляющий элемент со всеми своими основными и вспомогательными окнами способен работать с любой версией языка высокого уровня управляющих программ.

Второй уровень абстракции развязывает ActiveX-элемент и механизм управления им. Это позволяет использовать ActiveX или в составе терминальной задачи системы ЧПУ, или в отдельном приложении на персональном компьютере.

Заключение

Терминальная задача относится к числу наиболее сложных и наиболее ответственных разделов системы ЧПУ при управлении мехатронными системами. Ее «скелетом» служит интерпретатор диалога оператора в Windows-интерфейсе, для разработки которого использована формальная методика, поддержанная оригинальной инструментальной системой. Для редактирования, отладки и моделирования управляющих программ применяют два типа конфигурируемых приложений для управляющих программ низкого и высокого уровня соответственно.

3.5. Реализация диагностической задачи управления

Современные системы ЧПУ располагают определенными свободными ресурсами вычислительной мощности, которыми необходимо эффективно воспользоваться. Наиболее перспективный способ использования этих ресурсов состоит в создании и развитии подсистемы диагностики, которая в существующих системах представлена весьма слабо. В первую очередь следует диагностировать логическую и геометрическую задачи управления. Представлена концепция виртуальных приборов, построенных по типу ActiveX-элементов, которые позволяют использовать разработанные средства диагностики в различных приложениях, представляющих наибольший интерес для конечных пользователей. Особенности использованного далее COM-подхода и COM-технологии таковы, что разработанные диагностические системы могут быть применены в любых устройствах ЧПУ.

Наиболее совершенные системы ЧПУ располагают отдельным режимом диагностики, который реализован в виде программно-аппаратного комплекса и ориентирован на тестирование и глубокое исследование логической и геометрической задач управления. Диагностика, как правило, выполняется «вне реального времени», что означает – измерения сохраняются в памяти, а затем анализируются. Подсистема диагностики способна конфигурировать измерения, считывать измеряемые сигналы, запоминать результаты измерений вместе с результатами конфигурации измерений, распечатывать осциллограммы измерений, считывать файлы с результатами измерений и результатами конфигурации измерений, выполнять разнообразные операции над измеренными сигналами. Для диагностики логической задачи управления служит логический анализатор, а для диагностики геометрической задачи предназначен осциллограф.

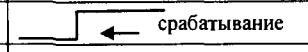
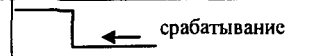
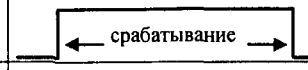
3.5.1. Понятийный аппарат диагностического процесса

К числу важнейших в практике диагностических измерений относятся такие понятия, как триггер, точка измерения, сигнал, состояние процесса измерения, виртуальный прибор диагностики.

Триггеры используют для формирования события, устанавливающего границы измерительного горизонта. Разнообразные их варианты представлены в табл. 9.

Группа стартовых триггеров устанавливает начало измерения, а окончание измерения определяет группа конечных триггеров. Кроме того, существуют триггеры специального назначения, например для выделения в

Таблица 9. Основные типы триггеров

Тип триггера	Условия срабатывания	Комментарий
Ручной (manual trigger)	Действие оператора	Нажатие кнопки оператора
Битовый (bit trigger)	По левому фронту	
	По правому фронту	
	По изменению значения	
BWD (байт, слово, двойное слово - Byte, Word, Dword trigger)	= (равно) <= (больше или равно) => (равно или меньше) != (не равно) < (больше) > (меньше)	Триггеры сопоставлены физическим адресам и срабатывают при выполнении логических условий
Программный триггер (program trigger)	Выполнение кадра УП	Задают номер кадра управляющей программы, при выполнении которого триггер срабатывает

процессе измерения некоторого события. Группа срабатывает при выполнении логических операций над ее триггерами.

Точки измерения представляют собой адреса аппаратных средств, осуществляющих измерительный процесс. Сигналы являются результатами измерительного процесса. Процесс измерения включает несколько фаз, которые назовем состояниями измерительного процесса; в числе возможных состояний – «конфигурация», «начало измерения», «ожидание», «конец измерения», «ошибка».

Под виртуальным прибором диагностики будем понимать ActiveX-элемент, предоставляющий результаты диагностических испытаний и создающий внешний образ измерительного устройства, например логического анализатора или осциллографа.

3.5.2. Структура подсистемы диагностики

Подсистема диагностики построена по типу виртуальной машины [63] и имеет многоуровневую структуру (рис. 103).

Нижний уровень представлен компьютерной аппаратурой, которая является физическим источником измерительных сигналов; здесь, в частности, могут быть программируемый контроллер, плата управления следящими приводами и т.д. Выше размещаются драйверы устройств ввода-вы-

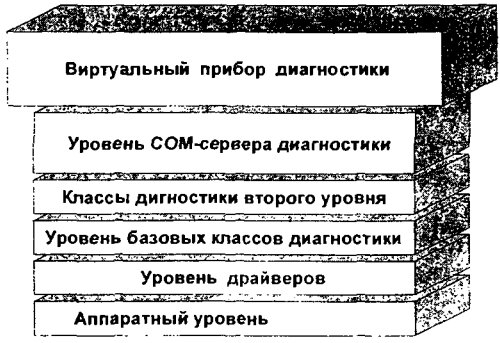


Рис. 103. Виртуальная модель подсистемы диагностики

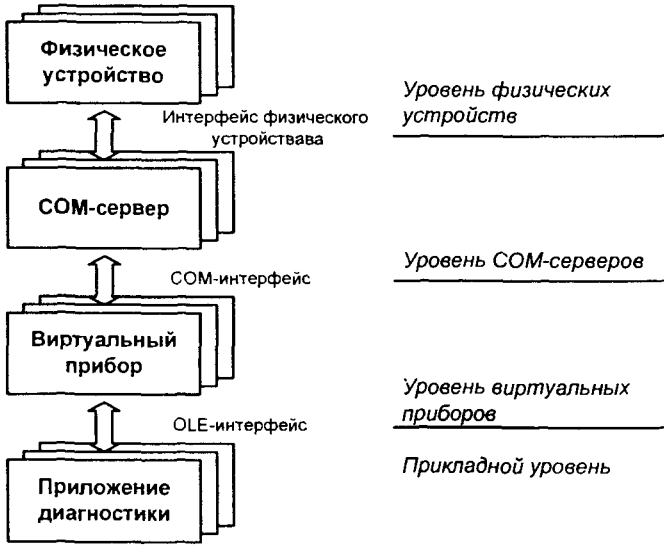


Рис. 104. Распределенная архитектура подсистемы диагностики

вода, которые входят в состав операционной системы. Доступ к службам устройств ввода-вывода осуществляется посредством слоя базовых классов, реализующих обмен данными с подсистемой диагностики, их форматирование и контроль. Поверх располагаются классы второго уровня, запускающие и контролирующие процессы измерения. Уровень COM-сервера [64] стандартизует доступ к подсистеме диагностики, с одной стороны, и поддерживает распределенную модель измерительной системы, с другой стороны. Третий – пятый уровни выстраивают объектную модель под-

системы диагностики, которая облегчает создание и поддержку пользовательских приложений.

Виртуальный прибор диагностики (на шестом уровне) предназначен для доступа оператора к результатам измерений [65]. Он подключается к интерфейсам COM-сервера диагностики и привязан к формату (типу) интерфейсов, а не к самой реализации COM-сервера. Это позволяет использовать его в разных системах управления.

В обобщенном виде архитектура подсистемы диагностики предстáвлена на рис. 105. Архитектура должна быть распределенной, чтобы удовлет-

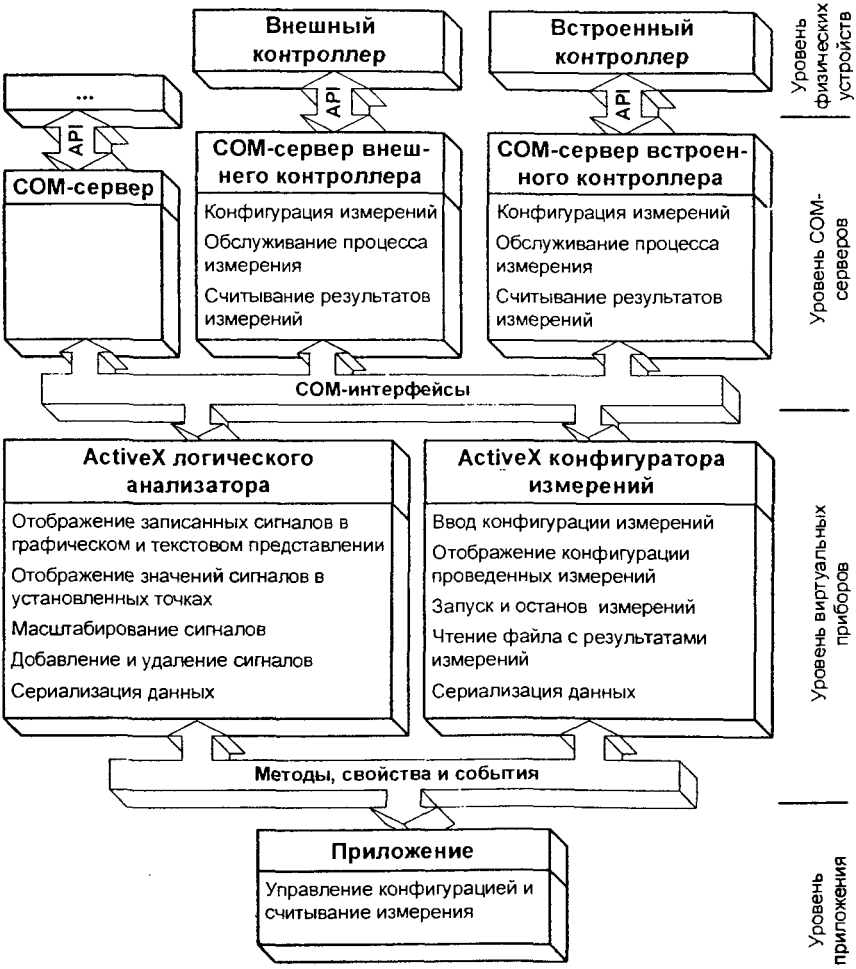


Рис. 105. Распределенная архитектура логического анализатора

ворить требованиям удаленной диагностики. Соединение с физическим устройством (контроллером ввода-вывода, контроллером приводов и т.д.) осуществляется с помощью интерфейсной функции этого устройства. COM-серверы маскируют особенности физических устройств, но организуют доступ к устройствам по общему COM-интерфейсу.

Уровень виртуальных приборов предлагает средства интерактивного конфигурирования и визуализации измерений. На прикладном уровне эти приборы встроены в приложение с доступом к приборам через OLE-интерфейс.

Систему, построенную на основе виртуальной модели подсистемы диагностики, назвали *логическим анализатором.*

3.5.3. Реализация логического анализатора

Электроавтоматика мехатронных систем достаточно сложна и требует высококвалифицированных специалистов при наладке и запуске оборудования в эксплуатацию. Подобные специалисты находятся обычно в удаленных сервисных бюро и, располагая конфигурацией и результатами измерений, занимаются дистанционным анализом входных и выходных сигналов программируемого контроллера с помощью все того же виртуального прибора.

Распределенная архитектура подсистемы диагностики, представленная на рис. 105, ориентирована на работу с внешними программируемыми контроллерами, а также с встроенными в систему управления. Функции компонентов архитектуры показаны на рис. 106.

Система диагностики программируемого контроллера построена на основе COM-сервера, в котором специфицированы пять интерфейсов

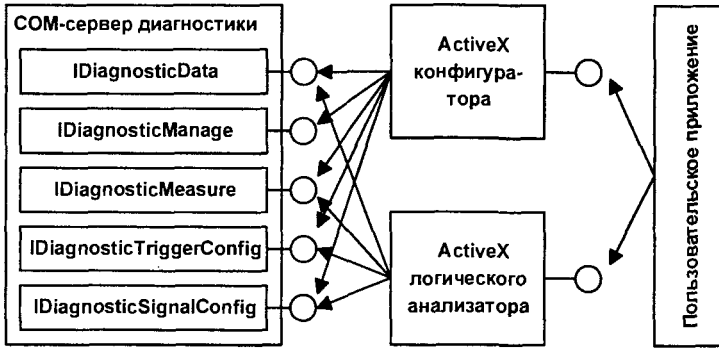


Рис. 106. Компонентная модель диагностики программируемого контроллера

- IDiagnosticData – интерфейс измерительных данных стандартных типов (BIT, BYTE, WORD, DWORD);
- IDiagnosticManage – интерфейс управления процессом измерений;
- IDiagnosticMeasure – интерфейс выдачи параметров результатов измерений;
- IDiagnosticTriggerConfig – интерфейс конфигурации условий запуска-окончания измерений;
- IDiagnosticSignalConfig – интерфейс конфигурации точек измерений.

Каждый из интерфейсов имеет неизменный набор функций. Сами интерфейсы реализованы как вложенные классы (nested classes) в классе CDiagnosticServer.

Любой виртуальный прибор реализован как ActiveX-элемент [66] и может быть встроен в стандартный или пользовательский контейнер в среде MS WindowsNT.

ActiveX конфигуратора предназначен для конфигурационных настроек (рис. 107). Конфигурация измерения может быть сохранена, отредактирована или практически использована при измерении. Измеренные данные могут быть прочитаны и отображены ActiveX логического анализатора (рис. 108). При просмотре отображаемые сигналы можно масштабировать, сравнивать между собой и т.д.

При работе подсистемы диагностики пользовательское приложение, как правило, не замечает COM-сервера диагностики, оно взаимодействует с ActiveX-элементом диагностики при помощи механизма OLE Automation.

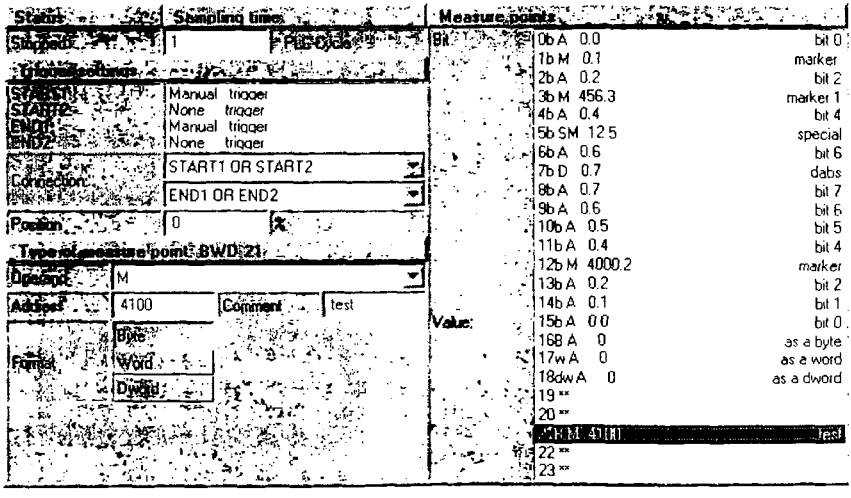


Рис. 107. Виртуальный прибор конфигурации измерений для программируемого контроллера

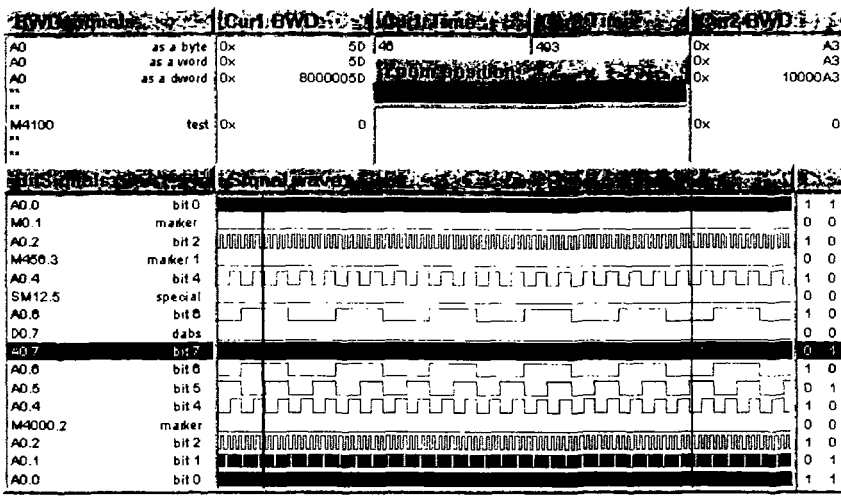


Рис. 108. Виртуальный прибор считывания измерений программируемого контроллера

ActiveX-элементы диагностики принимают на себя всю работу с сервером диагностики. Таким образом, функции клиента в основном остаются прозрачными для пользователя. От клиента спрятаны создание COM-сервера, получение указателей на интерфейсы, вызов функций этих интерфейсов, контроль за выполнением запросов и т.д.

Компонентная модель позволяет использовать пользовательское приложение для диагностики разных систем управления без перекомпиляции исходного кода. Для этого необходимо лишь разработать новый COM-сервер с учетом специфики новой системы управления, но с прежними интерфейсами.

3.5.4. Реализация осциллографа

Оптимальная настройка регуляторов следящих приводов подачи невозможна без тщательного анализа их динамических характеристик с помощью осциллографа подсистемы диагностики. Особенность распределенной архитектуры осциллографа (рис. 109) состоит в использовании «процесс-COM-сервера», в котором собраны все операции над сигналами, независимо от устройства-источника этих сигналов. В числе возможных операций над сигналами такие, как: масштабирование, сдвиг, практически любые математические вычисления.

Компонентная модель диагностики следящих приводов использует следующие интерфейсы:

- IOscManage для управления процесс-сервером, использует методы работы с внутренней базой данных объектов;

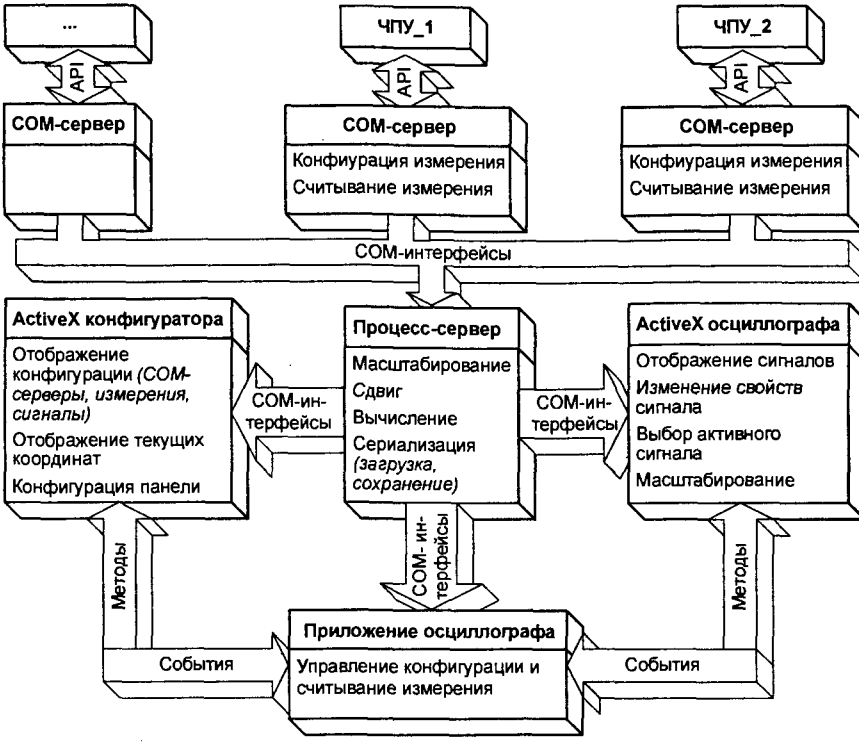


Рис. 109. Распределенная архитектура осциллографа

- IOscMeasureConfig для конфигурирования измерений, использует методы создания, удаления, установки триггеров, установки и считывания свойств триггеров;
- IOscMeasureData для работы с измерениями, содержит методы добавления, удаления, запуска и остановки измерений, а также условия срабатывания ручного триггера;
- IOscProcDataLoad для ввода-вывода исходных данных, содержит методы добавления и удаления серверов физических устройств, загрузки и сохранения данных и конфигурационных настроек;
- IOscScaling для управления отображением сигналов, содержит методы масштабирования и сдвига сигнала или группы сигналов, методы добавления и удаления сигнала из группы;
- IOscSignalData для работы с сигналами, содержит методы установки и считывания свойств и значений сигналов;
- IOscVisObjData для работы с визуальными объектами (сетками, курсорами и координатными осями), содержит методы добавления и удаления визуальных объектов, получения данных для этих объектов;

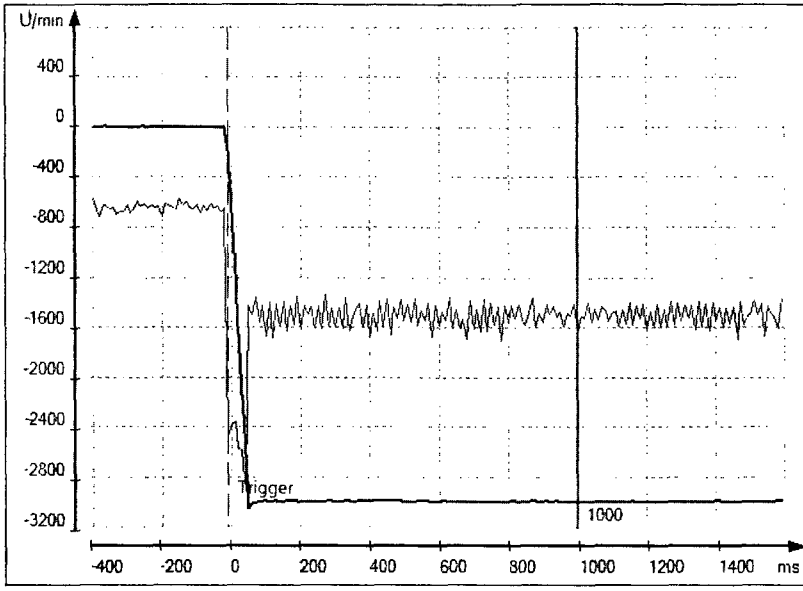


Рис. 110. Виртуальный прибор отображения данных в графическом виде

• IOscWindowData для управления параметрами отображения данных, содержит методы установки и считывания ширины и высоты области отображения.

Виртуальный прибор ActiveX конфигуратора работает в двух режимах: конфигурации измерения и отображения измерительных данных в текстовом формате. Виртуальный прибор ActiveX осциллографа отображает сигналы в графическом виде (рис. 110). ActiveX имеет возможности визуальной настройки свойств – цветов, шрифтов, стилей изображения сигналов. Приложение осциллографа подсистемы диагностики, помимо стандартных процедур конфигурации и отображения измерения, позволяет строить с помощью процесс-сервера амплитудно-частотные и фазочастотные характеристики следящих приводов (рис. 111).

Заключение

Современные системы управления располагают свободными ресурсами вычислительной мощности, которые должны быть использованы наиболее эффективно. В этом смысле наибольший интерес представляют создание и развитие подсистемы диагностики. В первую очередь следует диагностировать логическую и геометрическую задачи управления. Концепция виртуальных приборов, построенных по типу ActiveX-элементов, позволяет использовать разработанные средства диагностики в самых

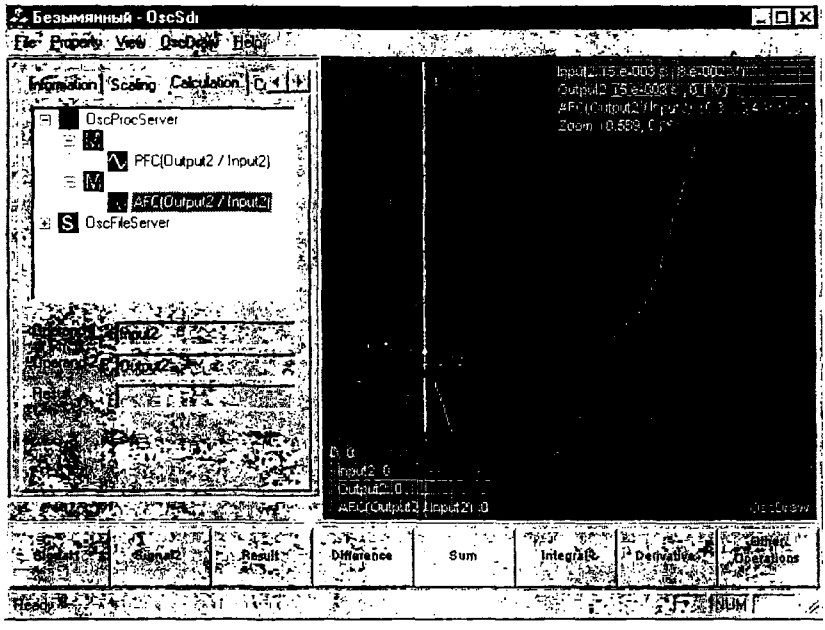


Рис. 111. Амплитудно-частотная характеристика в логарифмической системе координат (Бодс-диаграмма)

различных приложениях, которые представляют наибольший интерес для конечных пользователей. Особенности COM-подхода и COM-технологии таковы, что разработанные диагностические системы могут быть применены в любых устройствах ЧПУ.

Глава 4.
Технологии разработки программного обеспечения систем управления

Разработка систем управления предполагает привлечение целого ряда информационных технологий, использующих объектно-ориентированный и компонентный подходы, Windows-программирование, параллельное программирование и программирование систем в реальном времени. Но знание таких технологий является лишь необходимым условием для успешной разработки системы. Существует специфика использования этих технологий в системах управления. В частности, нужно хорошо представлять разницу между разработкой программного обеспечения офисного приложения и программного обеспечения систем управления, поскольку то и другое построено на разных моделях.

Если фирма Microsoft может позволить себе публикацию нескольких обновлений каждую неделю и выпуск SP (Service Pack) раз в полгода, то обновление программного обеспечения систем управления означает реализацию новой версии с одновременной поддержкой всех предыдущих версий. И если обновления офисных программ вполне реально выполнять через Internet, то для систем управления промышленным оборудованием такой возможности не существует.

4.1. Технология объектно-ориентированного программирования

Традиционное математическое обеспечение систем числового программного управления до сих пор использует технологию процедурного программирования, которую сегодня можно посчитать тупиковой в этой конкретной области применения. На смену приходят технологии объектно-ориентированного и компонентного программирования. В этой связи применительно к системам управления рассмотрены базовые понятия объектно-ориентированного подхода, представлены методические рекомендации по выбору объектов в системе управления, проанализированы проблемы создания объектно-ориентированных моделей, в том чис-

ле на основе формализма Г. Буча, и проблемы повторного использования уже разработанных кодов, продемонстрирована возможность инструментальной поддержки объектно-ориентированного программирования.

Методология открытой архитектуры приоткрывает конечному пользователю системы ЧПУ определенную часть математического обеспечения, с тем чтобы внедрить в систему свое собственное технологическое «know how». Поэтому разработчики математического обеспечения обязаны сделать его в достаточной степени обозримым и читаемым. Но это важно и самим разработчикам, чтобы внести в свой проект элементы структуризации и четкой организации, поскольку только на подобной основе огромный по объему проект может эволюционировать и совершенствоваться [67].

4.1.1. Сравнение технологий программирования

Модель процедурного подхода предполагает набор процедур, вызывающих друг друга для обработки данных. Инженер объясняет программисту проблему, а тот в свою очередь кодирует ее набором данных и процедур. Полученную программную систему сложно отладить и запустить, но еще сложнее изменить и расширить.

В основе объектно-ориентированного подхода лежит объект, имеющий определенные свойства (данные) и располагающий возможностями работы с этими данными. Модель объекта близка системе естественных физических представлений и способу мышления инженера, который оперирует привычными для себя понятиями. Поэтому и программная модель вполне адекватна физической. В результате объектно-ориентированный подход породил специалистов, объединяющих специальные инженерные и программистские знания.

Компонентный подход появился в результате дальнейшего развития объектно-ориентированного подхода, при котором наиболее важным фактором стала фиксация интерфейсов между компонентами программной системы при достаточно свободной реализации самих компонентов. Подобная идея также интуитивно соответствует классическому инженерному мышлению.

В рамках сравнения технологий программирования чрезвычайно важно рассмотреть проблему повторного использования уже разработанного кода. Процедурный подход решает эту проблему посредством динамически подключаемых библиотек DLL (Dynamic Link Library), реализованных в двоичном коде. Для библиотек DLL возможно использование разных языков программирования, однако существует проблема несовместимости версий одноименных библиотек. Есть и другой недостаток: отсутствует стандартный способ установки в системе более чем одной версии той же библиотеки.

Повторное использование объектов означает открытое представление исходных кодов, что не всегда соответствует авторским интересам разработчиков. Есть и другие проблемы: необходимость работать с одним и тем же компилятором; невозможность применения объектов в разных языках программирования; необходимость перекомпилирования всех проектов, использующих тот класс, который подвергся изменению.

Повторное использование компонентов составляет саму суть подхода. Реализация компонентов в двоичном коде соблюдает авторские права разработчиков, а стандартные интерфейсы делают компоненты общедоступными и способствуют их широкому (коммерческому) распространению. Работа компонентов поддерживается самой операционной системой.

Таким образом, можно заметить, что процедурный подход не ориентирован на многократное использование уже разработанных программных кодов; процедурные модели, как правило, понятны только программистам. Решение обозначенных выше проблем следует искать в применении объектно-ориентированного и компонентного подходов. Рассмотрим прежде объектно-ориентированный подход, поскольку он появился раньше компонентного и послужил для него основой.

4.1.2. Базовые понятия объектно-ориентированного подхода

Нередко в литературе одни и те же понятия объектно-ориентированного подхода обозначаются различным набором терминов из разных объектно-ориентированных языков, развивавшихся параллельно. Примем в дальнейшем некоторое соглашение по поводу терминологии.

Пять концептуальных понятий определяют объектно-ориентированную технологию: объект, класс, инкапсуляция, полиморфизм и наследование [68]. Объект содержит набор данных и методов. Данные (атрибуты) определяют специфические характеристики объекта, связанные с его назначением, в то время как методы ограничивают набор действий, которые объект способен совершить. Объект определяется с помощью класса, который обобщает, конкретизирует и специфицирует все свойства объекта. Например, класс «Привод» (Drive) содержит данные о состоянии, режиме и аварийных ситуациях координатного привода подачи, а также набор методов, включая процедуры запуска, останова и обработки нерегулярных ситуаций. Конкретным объектом привода является экземпляр класса «Привод».

Объявление класса на языке C++ выгладить следующим образом:

```
class CDrive
{
};
```

Добавим поля данных (m-поля: m_State, m_Mode), характеризующих состояние и режим работы привода:

```
class CDrive
{
 unsigned long m_State;// Состояние привода
 unsigned long m_Mode;        // Режим привода
};
```

Инкапсуляция защищает данные объекта от несанкционированного доступа и модификации. С помощью инкапсуляции данные объекта могут быть изменены только посредством собственных методов объекта; прямой доступ к данным объекта невозможен для клиента объекта. Например, чтобы разрешить доступ к начальному (пусковому) сигналу привода, инкапсуляция вынуждает клиентское программное обеспечение использовать специальные методы запуска и останова, которые исключают некорректное управление приводом.

Добавим методы (Start, Stop), реализующие функциональные возможности привода:

```
class CDrive
{
 unsigned long m_State;// Состояние привода
 unsigned long m_Mode;        // Режим привода
 public:
 bool virtual Start();      // Запуск привода
 bool virtual Stop();       // Останов привода
};
```

Полиморфизм означает, что разные объекты могут быть интерпретированы как одинаковые для одного и того же (своего) клиента, хотя их поведение различно. Допустим, что существуют класс аналогового привода и класс цифрового привода со своими методами запуска для каждого класса. Для клиентского программного обеспечения отсутствует разница в запуске аналогового и цифрового приводов, хота по сути процедуры запуска абсолютно различны.

Полиморфизм скрывает специфику реализации объектов от клиентского программного обеспечения, удерживая клиентов в неведении относительно тех деталей реализации объектов, которые не имеют к клиентам прямого отношения. Таким образом, он резко упрощает разработку клиентского программного обеспечения.

Наследование позволяет объекту производного (порожденного) класса автоматически наследовать при создании объекта данные и методы существующего родительского класса. Допустим, что имеется обобщенный базовый класс «Привод». Производный от него класс – «Цифровой привод» –

наследует специфические данные и методы своего родителя. Наследование позволяет повторно использовать ключевые части другого объекта, что существенно сокращает время разработки. Функциональные возможности, полученные в результате наследования, нередко должны быть скорректированы или расширены. Поэтому в производном классе переписывают (переопределяют) те функции, которые меняют поведение класса. Множество унаследованных классов образует иерархию (дерево) наследования.

Унаследуем класс «Цифровой привод» от существующего базового класса, переопределим функции запуска и останова и добавим функцию настройки (Adjust) привода посредством некоторого набора параметров pParamStruct:

```
class CDigitalDrive: public CDrive
{
 public:
 bool virtual Start();        // Запуск привода
 bool virtual Stop();              // Останов привода
 bool virtual Adjust(void* pParamStruct); // Настройка
};
```

Одиночное наследование заставляет программиста выбирать между двумя практически равноценными классами. В этой связи привлекает множественное наследование, доступное в языке C++, когда один класс использует структуру и поведение других классов. Однако множественное наследование может стать источником проблем (неопределенностей, конфликтов), когда родительские классы имеют общий прародитель с виртуальными функциями или элементы (данные или методы) с одинаковыми именами. На рис. 112 класс D унаследован одновременно от классов B и C, которые имеют общий прародитель – класс A.

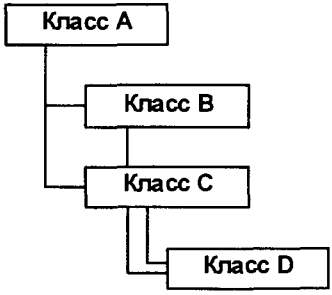


Рис. 112. Иерархия классов

Жизненный цикл объекта начинается с момента его создания, т.е. отведения участка памяти, и завершается возвращением системе этого участка памяти. Метод создания объекта называют конструктором, а метод освобождения объекта – деструктором.

Добавим конструктор по умолчанию (т.е. без параметров) и виртуальный деструктор:

```
class CDrive
{
 CDrive();                    // Конструктор
 virtual ~CDrive();           // Деструктор
```

```
unsigned long m_State;        // Состояние привода
unsigned long m_Mode;              // Режим привода
public:
bool virtual Start();         // Запуск привода
bool virtual Stop();          // Останов привода
};
```

Класс «виден» через свой внешний интерфейс, который и служит для его объявления. Области public, protected и private обозначают поля данных и методы, объявленные в этих областях, соответственно, как общедоступные всем клиентам, доступные только для унаследованных классов, доступные лишь для данного класса.

Иногда для построения сложного механизма удобно объявить некоторую функциональную возможность (метод) в родительском классе, а реализовать эту возможность в производных классах. Виртуальный метод, объявляющий эту функциональную возможность, называют абстрактным (pure method, чистый метод), а класс, содержащий хотя бы один абстрактный метод, – абстрактным классом, поскольку он не может иметь собственных объектов. Добавим абстрактный метод SelfTest в класс «Привод» для выполнения самотестирования:

```
class CDrive
{
 unsigned long m_State; // Состояние привода
 unsigned long m_Mode; // Режим привода
 public:
 bool virtual Start(); // Запуск привода
 bool virtual Stop(); // Останов привода
 protected:
 long virtual SelfTest(long Index) = 0;
 // Выполнение одного из внутренных тестов
};
```

Рассмотрим механизм виртуальных функций (методов), на котором построен полиморфизм объектов. Виртуальная функция (ее еще называют обобщенной) может быть переопределена в производных классах. Вызов виртуальной функции осуществляется не непосредственно по указателю, как для обычной функции, а через ее адрес в таблице виртуальных функций. Таблица привязана к объекту, ее первый элемент служит указателем на сам объект (указатель this), далее следуют адреса виртуальных методов в порядке их объявления.

Если производный класс переопределяет виртуальную функцию, то в таблицу виртуальных функций его объектов вносится адрес переопределенной, а не родительской функции. Все вновь объявленные виртуальные

Таблица виртуальных функций объектов класса CDrive
CDrive::this
CDrive::~CDrive()
CDrive::Start()
CDrive::Stop()
CDrive::SelfTest()

Таблица виртуальных функций объектов класса CDigitalDrive
CDigitalDrive::this
CDrive::~CDrive()
CDigitalDrive::Start()
CDigitalDrive::Stop()
CDigitalDrive::SelfTest()
CDigitalDrive::Adjust()

Рис. 113. Таблицы виртуальных функций объектов родительского и производного классов

функции располагаются в таблице вслед за родительскими (рис. 113). Существует полезное правило, которое гласит: если есть сомнения, делать ли метод виртуальным, лучше сделать его виртуальным.

Существует полезная категория параметризованных классов, которые в языке C++ называют шаблонами (templates). Они обобщают операции над разнородными элементами множества. Классический пример шаблона – это класс, выполняющий операции над коллекциями с тем, чтобы добавлять, удалять, заменять элементы или сортировать их по какому-либо признаку.

При разработке проектов чрезвычайно удобна иерархическая стандартная библиотека классов MFC (Microsoft Foundation Classes), которая предлагает уже готовые программные решения. Вершиной иерархии служит класс CObject; который, за некоторым исключением, порождает остальные классы библиотеки.

Можно построить целую группу классов, работающих с объектами CObject как с коллекциями, создавая массивы, связанные списки и карты. В библиотечном классе CString хорошо формализовано использование строк. Все оконные классы MFC, включая классы стандартных управляющих элементов Windows, производны от класса CWnd. Они предлагают широкие возможности по отладке, обработке исключений (ошибок), форматированию данных и др. Методика использования MFC при создании программного обеспечения систем управления заслуживает отдельного рассмотрения, и это предполагается сделать в дальнейшем.

4.1.3. Методические рекомендации по выбору объектов в системе управления

Объекты следует выбирать соответственно реальным физическим и логическим элементам системы управления. Например, объектами системы ЧПУ могут быть приводы координатных осей, каналы управления, входы

и выходы системы управления электроавтоматикой, режимы управления системой ЧПУ, экраны интерфейса оператора, отдельные модули системы, буферы обмена данными и т.д.

Данные объекта должны соответствовать данным, характеризующим моделируемый физический или логический элемент. Например, привод координатной оси определяется своим уникальным идентификатором в системе, именем, статическим состоянием (работает в системе или нет), типом (линейный или поворотный), текущим значением координаты, максимальным и минимальным ограничителями и т.д. (рис. 114).

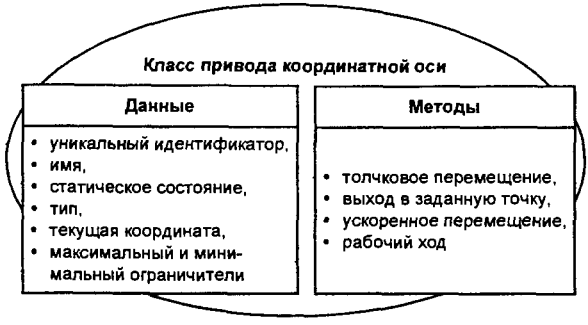


Рис. 114. Пример прототипа класса

Методы привода координатной оси устанавливают такие функциональные возможности, как толчковое перемещение, выход в заданную точку, ускоренное перемещение, рабочий ход и т.д.

4.1.4. Структура программного обеспечения системы управления

Объекты системы управления играют троякую роль: 1) содержат данные, определяющие свойства объекта; 2) обладают некоторыми функциональными свойствами; 3) обеспечивают презентацию объекта в интерфейсе оператора [69]. При этом возможна проблема интеграции сложившихся объектов с разными модификациями программных пакетов, соответствующих, например, вариантам реализации электроавтоматики или привода подачи.

Новая концепция Microsoft Windows DNA (Distributed IntraNet Application – распределенное сетевое приложение) предлагает решение на основе трехуровневой архитектуры, в которую полностью интегрированы объекты системы управления (рис. 115). Windows DNA выделяет: клиент-

Рис. 115. Трехуровневая архитектура Windows DNA

скую часть, осуществляющую представление данных; серверную часть, поставляющую данные, и некоторый промежуточный слой, снимающий прямую зависимость клиентской части от данных.

Таким образом, можно изменять данные без прямого влияния на код уровня представления. Более того, промежуточный слой может реализовать логику управления различными клиентскими приложениями, что приведет к увеличению объема повторного использования кода. Межуровневый интерфейс использует компонентный подход, а сами уровни имеют объектно-ориентированную реализацию.

Внедрение Microsoft DNA технологии в системы управления создает платформу объектов, которая напрямую может быть использована для решения разнообразных других проблем, таких, как управление потоками материалов, контроль качества, планирование производства.

4.1.5. Инструментальная поддержка объектно-ориентированного проектирования и формализм Буча

Инструментальные CASE-системы (Computer-Aided Software Engineering) ускоренной разработки программного обеспечения построены на различных нотациях представлений объектно-ориентированных моделей. Одной из самых распространенных и удачных является нотация Буча [70, 71].

Фрагмент нотации Буча рассмотрим на примере диаграммы, которая отображает классы и их отношении, тем самым раскрывая логику проекта. Класс обозначают в виде облака, в котором представлены имя, атрибуты (данные) и операции (методы) класса. Тип видимости атрибута или операции отображается соответствующей пиктограммой.

Абстрактный класс обозначают буквой «А» в треугольнике. Наследование показано стрелкой, направленной от производного класса к родительскому. Линии с точкой на одном конце обозначают отношение между классами. Если это отношение включения (агрегации), то линия заканчивается квадратиком. Квадрат закрашен, если включение осуществлено по значению, и не закрашен, если включение осуществлено по ссылке. В качестве примера на рис. 116 в классе CDrive добавим поле m_Name, соответствующее имени привода. Стандартный класс CString библиотеки MFC обладает набором функций для хранения и работы со строковыми массивами, поэтому он был использован как тип поля m_Name.

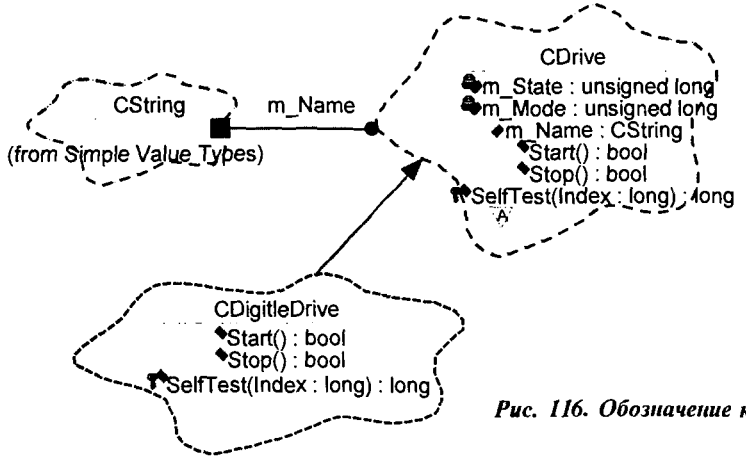


Рис. 116. Обозначение класса

Для полного описания объектно-ориентированной модели нотация Буча использует диаграммы:

- взаимодействия для описания взаимодействующих групп объектов;
- состояний и переходов для описания возможных состояний объектов и действий при изменении состояний;
- модулей для распределения классов и объектов по модулям;
- процессов для распределения задач по физическим устройствам и процессорам, обеспечивающих работу системы.

Диаграмма взаимодействия показывает объекты и сообщения, которыми они обмениваются между собой в рамках некоторого конкретного механизма, и позволяет проследить выполнение сценария работы механизма. Диаграммы взаимодействия полезны при описании поведения нескольких объектов в рамках одного варианта использования.

Диаграмма состояния определяет состояния, в которых может находиться конкретный объект; события, инициирующие переход из одного состояния в другое; действия, инициируемые изменениями состояний. Эти диаграммы используют для анализа поведения отдельных классов при их взаимодействии и динамики системы в целом.

Диаграмма модулей демонстрирует архитектуру системы в ее слоях и разделах. Набор модулей показывает структуру проекта. Модулю главной программы сопоставлен файл с расширением *.cpp, содержащий основную функцию main() или WinMain() для Windows. Модулю описания соответствует файл заголовков с расширением *.h, в котором объявляются клас-

сы. Модулю тело соответствует файл с расширением *.cpp, в котором представлены классы. Модулю подсистемы соответствует библиотека на языке C++. В подсистеме размещены логически связанные модули описания и модули тела.

Диаграмма процессов представляет физическую совокупность процессов и устройств, обеспечивающих работу системы.

Формализм Буча нашел свое отражение в популярной CASE-системе Rational Rose. Система предполагает применение формализма Буча на этапе анализа и проектирования объектно-ориентированной модели. На основе модели генерируются исходные C++ коды, после чего программисту остается придать сгенерированным классам необходимые функциональные возможности. Реинжиниринг (т.е. обратная генерация объектно-ориентированной модели из исходного кода) позволяет системе Rational Rose добавлять в модель изменения, внесенные в C++ файлы.

Заключение

Объектно-ориентированная технология создает на базе классов и объектов хорошо структурированное модульное программное обеспечение, улучшает сохранность данных путем инкапсуляции, уменьшает сложность клиентского программного обеспечения за счет полиморфизма и увеличивает надежность программного кода с помощью наследования. Технология объектно-ориентированного и компонентного подходов является единственно приемлемой при разработке сложного и объемного программного обеспечения систем ЧПУ мехатронными системами.

4.2. Специфика объектно-ориентированного программирования

Объектно-ориентированная модель системы PCNC структурирует архитектуру системы, делает программное обеспечение прозрачным и повышает, следовательно, его надежность, упорядочивает процесс разработки, создавая предпосылки для формирования его среды. Рассмотренные элементы базовых абстракций, классы их объектной реализации, построенные на основе этих классов механизмы служат основой для создания элементов системы ЧПУ более высокого уровня. К числу таких элементов относятся каналы, оси, закрепленные за этими каналами, и т.д.

Общая технология объектно-ориентированного программирования формирует универсальный подход к созданию программного обеспечения, но не учитывает специфику каждой реализации. Для раскрытия специфики объектно-ориентированного программирования системы ЧПУ типа PCNC

необходимы ее абстрактная модель, объектно-ориентированная модель обобщенного модуля, объектно-ориентированная модель отображения данных. Совокупность подобных моделей позволяет просмотреть ключевые фазы разработки программного обеспечения ЧПУ: анализ, проектирование, реализацию.

4.2.1. Элементы абстрактной модели системы PCNC

Абстрактная модель системы PCNC может быть представлена совокупностью данных, команд и процессов (рис. 117). В соответствии с объектно-ориентированным представлением каждый из элементов совокупности определяют наборами свойств и функциональных возможностей (функциональностей), причем содержательно функциональные возможности состоят в выполнении операций над элементами. Выделение элементов позволяет строить единообразные механизмы для системы в целом и ее

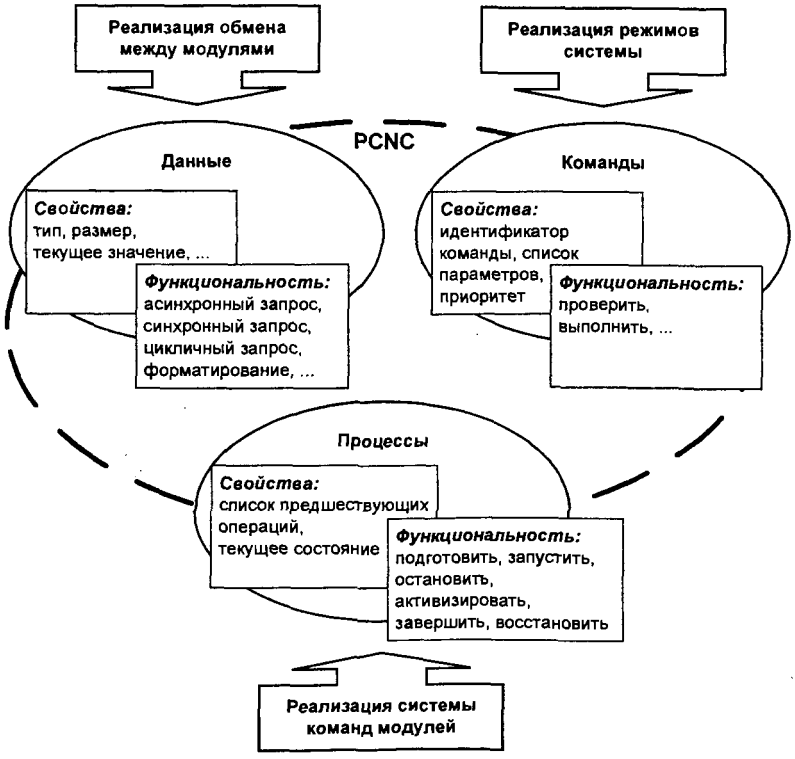


Рис. 117. Выделение элементов абстрактной модели системы PCNC

компонентов. Так, на базе элементов абстрактной модели выстроен межмодульный обмен данными, организованы базовые режимы, определена система команд PCNC. Характер межмодульных транзакций, т.е. сессий обмена данными, зависит от категории последних.

К первой категории отнесли те данные, без которых выполнение определенных операций попросту невозможно. Сюда относятся все данные, необходимые для расчетов. Например, без знания табличного значения радиуса инструмента невозможно рассчитать эквидистантную коррекцию. Для получения подобных данных используют синхронную транзакцию: модуль, запрашивающий данные, приостанавливает свою работу до их получения.

Во вторую категорию включили данные, использование которых в текущий момент некритично. Таковыми могут быть данные, привязанные к наступлению какого-либо события, на которое модуль должен определенным образом отреагировать. В этом случае модуль запрашивает данные асинхронным способом, формируя соответствующую заявку. Запросив их, он не прекращает своей работы, а поступающий ответ обрабатывается так называемой callback-функцией.

Данные визуализации (выводимые на экран) нуждаются в постоянном циклическом опросе (они отнесены к третьей категории). Для минимизации объема транзакций используют асинхронный запрос по событию. Событием в этом случае служит изменение отслеживаемых данных. После наступления события они передаются для обновления визуализации. Запрашивающий модуль регистрируется при этом как получатель («подписчик»).

Таким образом, категория данных устанавливает способ их запроса по виртуальной шине [38]. Вспомогательные категории определяют специальные способы запроса при отладке системы и ее программного обеспечения. Однако категория – не единственная характеристика данных абстрактной модели системы PCNC. Другими характеристиками являются их формат, размерность, корректность (как результат верификации), тип (char, integer, long, double, pointer, ...), объем занимаемой памяти и текущие значения.

Процессы абстрактной модели системы PCNC определены своими состояниями (активное, неактивное, приостановленное, состояние ошибки и т.д.), списком предшествующих операций, набором предусмотренных акций (инициализация, запуск, останов, завершение, выход из ошибочного состояния). Состояниями и акциями можно управлять. Управление акциями осуществляется синхронным или асинхронным способом (подобно транзакциям передачи данных). Понятие «процесс» близко к понятию «режим» системы ЧПУ (автоматический, ручной, толчковый, запуска строки ручного ввода и т.д.). Поскольку практически все модули системы ЧПУ зависимы от режимов, они естественным образом связаны с процессами.

Принцип построения системы PCNC по типу языкового процессора [2] предполагает, что в качестве команд выступают G-функции кода ISO-7bit. В терминах команд осуществляются анализ, интерпретация и отработка кадров управляющей программы путем последовательного конвертирования информации в форматы интерполятора и приводов. Каждая команда имеет свой идентификатор, список параметров и приоритет. Предусмотренные акции команд состоят в проверке возможности выполнения команд и их непосредственной реализации. Команды входят в сложные отношения между собой для образования комплексных иерархических структур. Например, набор команд интерпретатора определяется как система команд ISO-процессора [51], причем для организации параллельной интерпретации в составе базового набора команд выделяют группы, так называемые групповые интерпретаторы.

4.2.2. Объектно-ориентированная модель модуля системы PCNC

Выделение элементов абстрактной модели системы ЧПУ является базисом для построения объектно-ориентированных моделей механизмов системы PCNC. Иерархия классов системы PCNC построена с использованием основных принципов объектно-ориентированного подхода: инкапсуляции (возможности скрывать детали объекта от программиста), наследования (возможности создавать новые объекты на базе существующих), полиморфизма (возможности использовать различные варианты поведения объектов). В основе иерархии лежат классы, реализующие элементы абстрактной модели: данные, процессы, команды. Выделение этих классов поддерживает общие свойства и поведение производных (наследованных) классов и позволяет разработать с их помощью все основные механизмы системы PCNC.

Модуль системы PCNC строят по принципу командного процессора, на вход которого поступают команды с параметрами. Объектно-ориентированная модель модуля в нотации Буча [70] показана на рис. 118. Классы представлены в виде «облачков Буча», связанных между собой определенными отношениями. Ядром модуля служит языковой процессор, предназначенный для выполнения функциональных задач модуля. Языковый процессор соответствует классу CLanguageProcessor, производному от стандартного класса CObject библиотеки MFC. Класс наследует механизмы работы: с архивами – информацией на постоянных носителях, с RTTI (Run-Time Type Information) – информацией о типе, известном только во время выполнения приложения, с дампа памяти, и верификации в режиме отладки.

Прототип языкового процессора содержит систему команд, машину состояний и конфигуратор. Они соответствуют m-полям (member fields)

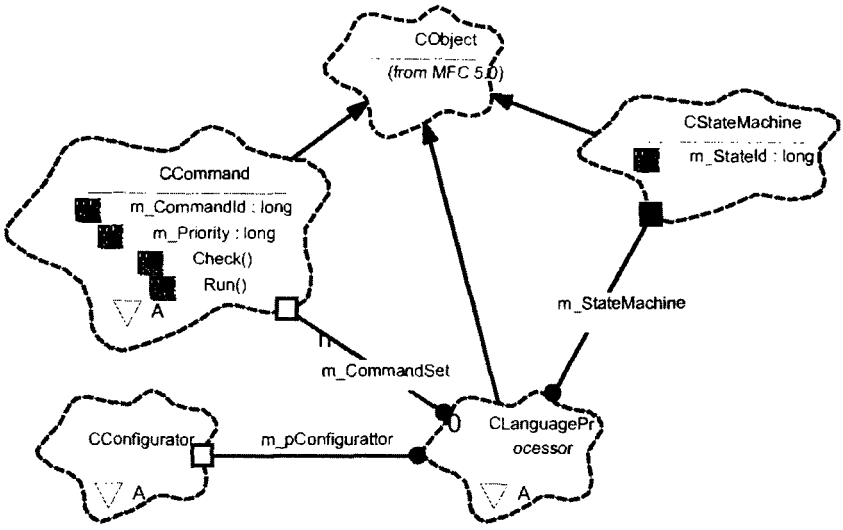


Рис. 118. Фрагмент диаграммы классов (в нотации Буча), реализующих абстрактный командный процессор

класса CLanguageProcessor: m_CommandSet, m_StateMachine и m_pConfigurator. Система команд реализована в виде набора классов, производных от CCommand, полученных в свою очередь от библиотечного класса CObject.

Класс CCommand имеет общие для всех команд свойства: идентификатор команды (поле m_CommandId), предназначенный для распознавания команды; приоритет команды (m_Priority). Если на вход процессора поступает одновременно несколько команд, последовательность их выполнения соответствует приоритетам. Реализации конкретных команд и их параметры определены классами, производными от CCommand. При этом переопределяются виртуальные функции CCommand::Check() – проверка на возможность выполнения, и CCommand::Run() – выполнение команды.

Класс CStateMachine, производный от CObject, следит за состоянием модуля. Состояние сохраняется в поле m_StatusId. В этом классе предусмотрены также функции проверки возможности перехода в новое состояние. Класс, производный от CConfigurator, предназначен для конфигурации, т.е. настройки на конкретный модуль. Конфигуратор включают в языковой процессор по указателю (т.е. поле m_pConfigurator содержит только ссылку на конфигуратор), что позволяет заменять его, т.е. работать с разными системами конфигурации.

4.2.3. Объектно-ориентированная модель отображения данных

Данные системы PCNC выделяют в классы, исходя из их неразрывности и функциональных характеристик. Объект, представляющий данные системы PCNC, включает в себя такие их характеристики, как тип, размер, текущие значения, а также функциональные возможности, позволяющие запрашивать данные разными способами, форматировать их, конвертировать и верифицировать и т.д. Практически все типы данных системы PCNC подпадают под описание абстрактного объекта CAbstractData (рис. 119). Примерами могут послужить текущее положение координатных осей, подача, кадр управляющей программы, состояние модуля. Конкретизация (уточнение) данных осуществляется в классах, унаследованных от CAbstractData, а механизм обмена данными реализуется на основе обобщенных функциональных возможностей этого класса.

Класс, поставляющий данные, служит сервером, а классы, получающие от него данные, являются его клиентами. В принципе клиент может быть описан некоторым абстрактным классом CReceiver, имеющим все-

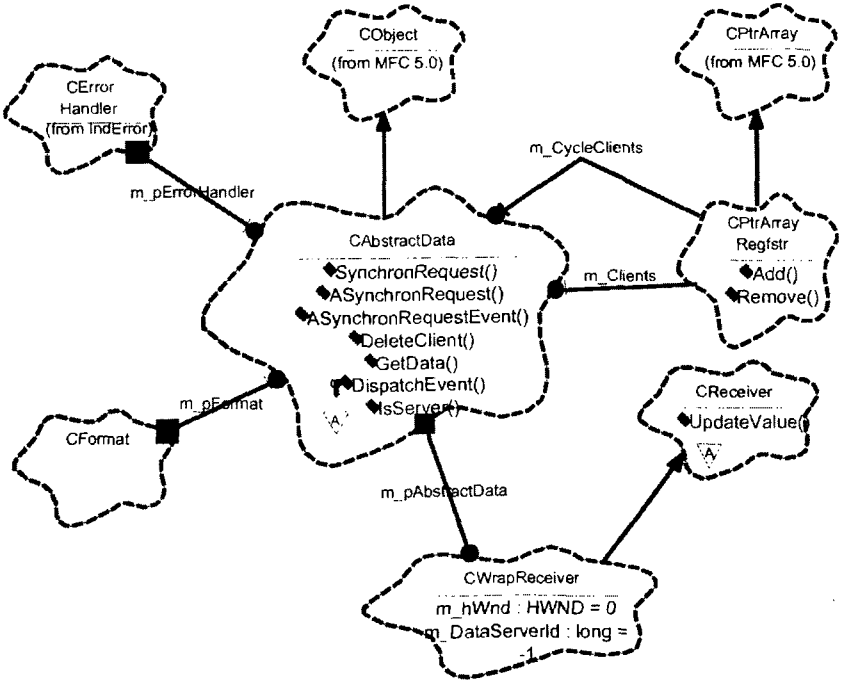


Рис. 119. Фрагмент диаграммы классов (в нотации Буча), реализующих механизм отображения данных

всего одну виртуальную функцию CReceiver::UpdateValue(). Эта функция класса и вызывается сервером для передачи данных. Клиенты, желающие получить данные от сервера асинхронным или циклическим способом, «подписываются» на них или их изменение. Подписка требует специального механизма регистрации клиентов со стороны сервера данных (рис. 120 и 121).

Класс AbstractData (см. рис. 119), поставляющий данные, имеет список m_Clients для асинхронных клиентов и список m_Cycle Clients для циклических клиентов. При любых запросах, обращенных к серверу, проверяется наличие клиента в списке. Если клиент не обнаружен, то он добавляется в соответствующий список функцией CPtrArrayRegistr::Add (см. рис. 120). При изменении данных клиент уведомляется посредством виртуальной функции CReceiver::UpdateValue(). После подтверждения достоверности данных клиент запрашивает их с помощью функции CAbstractData::GetData(). При

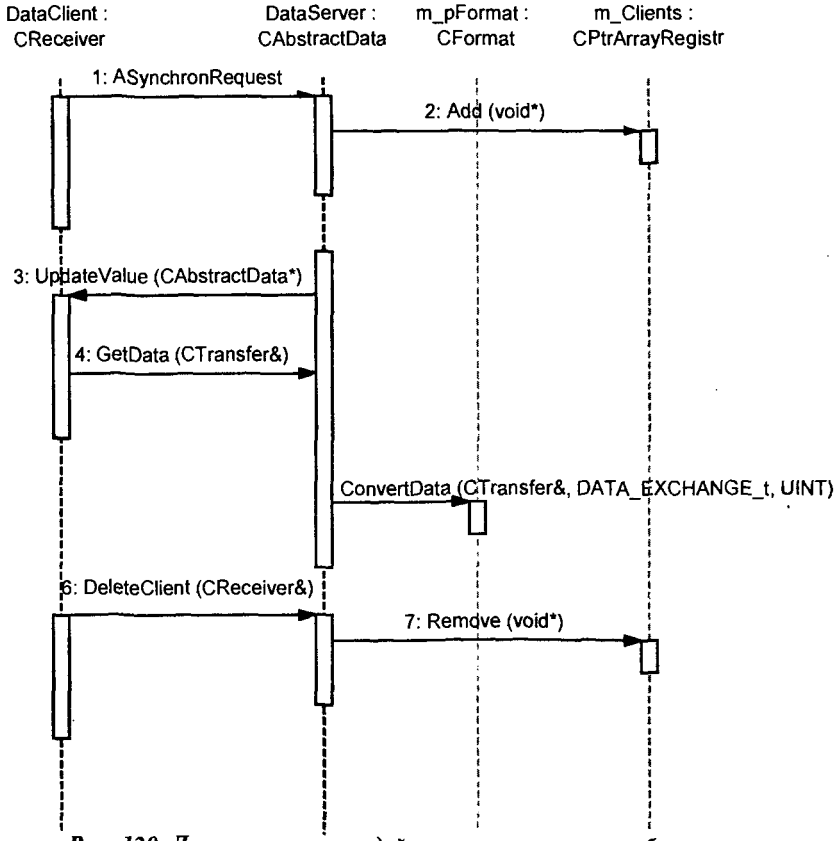


Рис. 120. Диаграмма взаимодействия механизма отображения

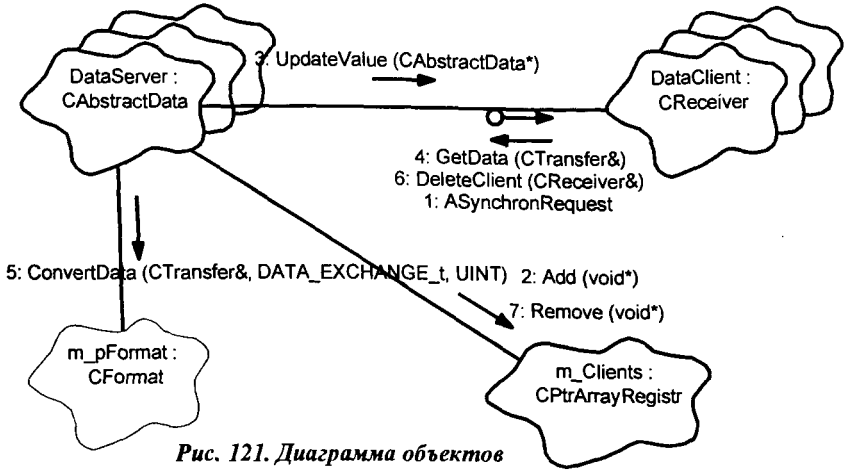


Рис. 121. Диаграмма объектов

этом каждый клиент определяет формат запрашиваемых данных. К примеру, текущие позиции координатных осей необходимы как в числовом виде для выполнения арифметических операций, так и в виде строк для отображения на экране. Функция CFormat::ConvertData() осуществляет конвертирование и форматирование данных. Асинхронные клиенты в списке m_Clients удаляются автоматически со стороны сервера после уведомления клиентов.

Циклический опрос прекращается со стороны клиента вызовом функции CAbstractData::DeleteClient(). Для описания сценария работы механизмов используют диаграммы взаимодействия и диаграммы объектов. Семантическая близость диаграмм позволяет использовать инструментальные средства генерации одной диаграммы из другой с минимальными потерями информации. Диаграмма взаимодействия более наглядна. Она похожа на таблицу, в верхней строке ее записывают имена объектов, под каждым из которых рисуют вертикальную штриховую линию. Горизонтальные линии соответствуют посылкам сообщений, их проводят от вертикали клиента к вертикали сервера и располагают в хронологической последовательности сверху вниз. Асинхронные сообщения обозначаются половинками стрелок.

Преимущества диаграммы объектов в том, что она более подходит для многих объектов со сложными вызовами и допускает включение дополнительной информации, например поток данных (стрелка с кружочком в конце) и т.д.

Опыт работы показывает, что диаграмма взаимодействия лучше передает семантику сценариев на ранних стадиях проектирования систем PCNC, когда протоколы взаимодействия отдельных классов не зафиксированы.

По мере того, как уточняется структура классов, вырисовываются подробности диаграммы объектов.

Разработанная нами объектно-ориентированная модель механизма отображения данных имеет пока одно ограничение – сервер и клиент работают с прямыми указателями друг на друга. Скорость такого обмена данными очень высока, но сам обмен возможен только в пределах единого адресного пространства, т.е. в пределах одного потока Windows NT. Класс CWrapReceiver, производный от CReceiver (см. рис. 119), решает проблему перехода границы потока путем разделения процедуры получения данных на две части:

- посылка клиенту уведомления из другого потока о достоверности данных;
- прямой запрос данных со стороны клиента и приведение этих данных к стандартному типу CSaferyArray.

Клиент подписывается на получение определенного типа данных асинхронным или циклическим способом, передавая при этом в качестве параметров идентификатор своего окна hWnd и идентификатор запроса данных. В соответствии с этими параметрами объект класса CWrapReceiver (оболочка) создается в потоке класса сервера данных и регистрируется в соответствующей очереди клиентов. При обновлении данных вызывается переопределенная виртуальная функция CWrapReceiver::UpdateValue().

Эта функция уведомляет клиента через границу потока о достоверности данных посредством стандартных механизмов Windows SendMessage() или PostMessage(). Оконная функция обработки сообщения клиента получает данные посредством стандартного типа CSaferyArray. Диаграмма объектов описанного механизма представлена на рис. 122. С помощью механизма отображения данных можно решать практически все задачи обмена данными в системе PCNC, в том числе слежение за возникновением ошибок.

Заключение

Объектно-ориентированная модель системы PCNC структурирует ее архитектуру, делает программное обеспечение прозрачным и повышает, следовательно, его надежность, а также упорядочивает процесс разработки, создавая предпосылки формирования среды разработки. Рассмотренные элементы базовых абстракций, классы их объектной реализации, построенные на основе этих классов механизмы, – все это служит основой для создания элементов системы ЧПУ более высокого уровня. К числу таких элементов относятся каналы, оси, закрепленные за этими каналами, и т.д.

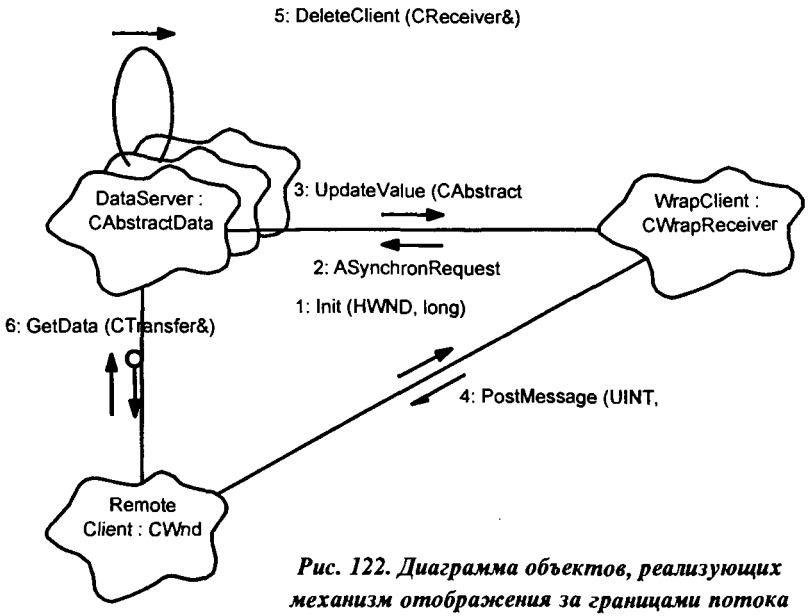


Рис. 122. Диаграмма объектов, реализующих механизм отображения за границами потока

4.3. Методологические аспекты построения открытых систем ЧПУ

Открытость систем ЧПУ типа PCNC должна быть передана в руки конечных пользователей, которые являются потребителями систем. Одним из вариантов решения проблемы является построение модулей системы ЧПУ по типу открытых языковых процессоров, что предопределяет регулярную архитектуру и конфигурируемую систему команд. Построение открытых языковых процессоров на базе объектно-ориентированного подхода решает проблему совместимости модулей. Ряд проблем разработки открытых систем PCNC может быть решен с помощью системных возможностей Windows NT. При этом возникает возможность использовать отлаженные решения и накопленный опыт, снизить затраты на разработку и повысить общую надежность системы.

Стандартные инструментальные средства должны быть также использованы для поддержания открытой архитектуры. Они формируют мощную подгруппу в окружении разработки систе-

мы PCNC, однако недостаточны. Поэтому для формализованных задач необходимо создавать собственный инструментарий. Формирование окружения разработки позволяет предложить комплексное решение проблемы создания открытой системы ЧПУ. При этом конкретные решения закрепляются за отдельными фазами процесса разработки таким образом, что охваченным оказывается весь итерационный процесс разработки.

Сегодня отсутствует методологическое решение проблемы построения открытых систем ЧПУ. Некоторые подходы обозначены проектами OSACA, OMAK, OSEC, реально же открытость систем остается в руках производителей. И это при том, что неизменным лозунгом производителей является открытость на уровне станкостроителей и конечных пользователей [72]. В силу консервативности производители систем ЧПУ не готовы к их кардинальным изменениям. В попытках придать открытость собственным системам преобладает элемент стихийности; производители, станкостроители и конечные пользователи различно трактуют сам термин «открытость».

Продавая открытую систему, производители предполагают, что пользователь способен доработать систему соответственно своим требованиям. Но не каждая станкостроительная фирма в состоянии содержать программистов, работающих на стыке системного и прикладного обеспечения систем управления. Возникает проблема методологических аспектов построения открытых систем. Речь идет о том, как формализовать хотя бы часть рутинной работы. Цель состоит в решении методологических проблем создания открытых систем PCNC, формировании единой интегрированной среды проектирования, разработке и поддержке проекта создания указанных систем. При этом интегрированная среда не должна препятствовать эволюции любого компонента и включению новых компонентов, в том числе оригинальных средств разработки.

Исследование методологических аспектов построения открытых систем PCNC потребовало решения таких задач, как формирование понятийного аппарата и представлений об открытых системах PCNC, разработка ключевых подходов к проектированию открытых систем PCNC и построению «окружения» процесса проектирования. При решении поставленных задач использовались: система PCNC по типу языкового процессора, стандартные средства операционной системы и инструментальных средств поддержания открытой архитектуры, оригинальные (вновь разработанные) средства поддержания открытой архитектуры. Конечная цель состоит в том, чтобы сделать процесс разработки системы PCNC регулярным с учетом специфики каждого экземпляра системы управления.

4.3.1. Понятийный аппарат открытых систем ЧПУ

Определим понятия, имеющие отношение к проектированию прикладной компоненты открытой системы PCNC. Самым общим является проект, содержащий информацию обо всех модулях системы PCNC и схеме их размещения в ней. Часто проект описывают dsw-файлом для MS Visual C++ и pjt-файлом для CASE-системы Rational Rose, но это не обязательно. К примеру, система MS Source Safe поддержки проектов хранит их в общей базе данных. Обычно модули системы PCNC представлены в виде подпроектов, что позволяет вести параллельную их разработку и сосредоточиться на отдельных функциональных особенностях.

Другим ключевым понятием является «документ». Он содержит данные, которые могут быть параметрами, характеризующими рабочий процесс, например текущие позиции осей. Документ сопоставлен модулю или каналу системы PCNC и хранит соответствующие значения данных. Последние могут быть представлены в различных видах. Представление (вид) документа PCNC системы – это экран интерфейса, где одни и те же данные могут иметь отображения в графическом или текстовом виде.

Приведем далее терминологию «окружения» проекта. В процессе разработки открытой системы PCNC выпускают версии для тестирования, адаптации у заказчика и т.д., которые называют релизами (releases). Релиз-версии оптимизированы по быстродействию и объему выполняемого кода. Кроме того, существуют отладочные (debug) версии проекта, которые содержат отладочную информацию. В случае обнаружения ошибки в проекте отладочная версия позволяет трассировать программы, отслеживать значения локальных и глобальных переменных, просматривать стек вызова функций и конкретную область памяти (dump) для выявления причины ошибки.

Маркетинговая политика породила новый способ распространения программного обеспечения – с отложенным платежом. Существует бесплатная shareware версия; как правило, это неполно функциональная версия продукта, которая позволяет лишь ознакомиться с возможностями приложения. Ключ полного доступа потребитель получает после полной оплаты. Shareware-версия для открытой системы PCNC может быть использована только по отношению к отдельным программным модулям, являющимся автономными законченными приложениями (например, редактор NC-кодов).

Документация проекта – важный этап процесса разработки, который, тем не менее, не является самоцелью [74]. Документация отражает архитектуру и реализацию системы или ориентирована на конечного пользователя и включает инструкции по установке и эксплуатации релизов. Документация должна эволюционировать вместе с проектом.

4.3.2. Представление о системе PCNC как об открытой системе управления

Посмотрим, как решается проблема открытости в существующих системах. Открытость в системе Andronic (Andron) подразумевает настройку интерфейса пользователя для поддержания его собственного технологического процесса. Подобная настройка осуществляется с помощью инструментального языка, который является, по сути, генератором специальных CAM-систем. Другие функции недоступны ни станкостроителям, ни конечным пользователям.

Открытость системы Typ3osa (Bosch) состоит в предоставлении пользователю широкого и открытого набора API-функций (Application Programming Interface). Система Sinumeric 840D (Siemens) предлагает некоторый способ доступа (сервис) к функциям ядра, который аналогичен принципу работы с многофункциональными прерываниями в DOS (например, 21 H) [75]. Открытость этих систем, таким образом, реализована лишь в слабой степени.

Помимо проблемы открытости, нерешенными остаются проблемы реализации многозадачности прикладной компоненты, контроля над ведением проекта, формализации рутинных работ при разработке и настройке системы под конкретного заказчика.

Когда мы говорим об открытости на уровне пользователя, то в первую очередь подразумеваем возможность использования вынесенных во входной язык API-функций, настройки с помощью конфигурационных файлов и файлов инициализации, изменения значения реестра Windows NT, добавления и синхронизации внешних приложений на базе входных и выходных файлов. И лишь во вторую очередь рассматриваем возможность получения сервиса от ядра системы CNC через API-интерфейс [73].

При систематизации средств построения открытых систем PCNC выделим четыре их группы, соответствующие решению ключевых методологических проблем:

- средства языкового процессора, отображающие часть API-функций во входном языке и конфигурационном файле модуля;
- стандартные операционные средства, предполагающие использование готовых объектов и механизмов, а также опыта решения проблем многозадачности;
- стандартные инструментальные средства комплексного ведения проекта, позволяющие структурировать и ускорять проектирование системы PCNC;
- оригинальные инструментальные средства, позволяющие частично формализовать процесс разработки и избавиться от рутинных операций, специфичных для процесса создания систем PCNC.

Рассмотрим эти группы более подробно.

4.3.3. Построение систем ЧПУ по типу открытого языкового процессора

Под открытым языковым процессором будем понимать модуль, у которого часть API-функций вынесена во входной язык или в файл конфигурации модуля, что позволяет использовать функциональные возможности ядра системы ЧПУ, не прибегая к прямому программированию. На рис. 123 система PCNC представлена набором языковых процессоров. На вход интерпретатора поступают управляющие программы; настройка на конкретную версию языка ISO-7bit осуществляется с помощью конфигурационного файла.

Выходом интерпретатора служит IPD-файл, содержащий команды интерполятору и программируемому контроллеру. Интерполятор обрабатывает свои IPD-команды и рассчитывает приращения приводов. Программируемый контроллер обрабатывает свои IPD-команды в соответствии с

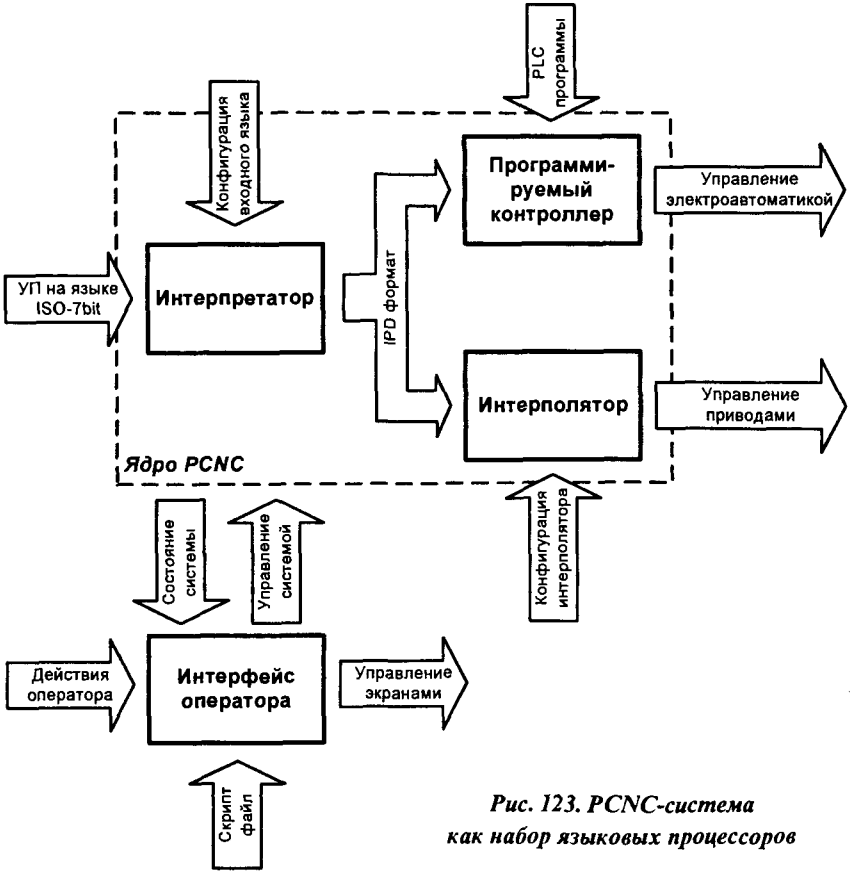


Рис. 123. PCNC-система как набор языковых процессоров

программами циклов и управляет электроавтоматикой. Экран показывает расположение управляющих элементов, размеры, шрифты, цвета. Язык выдачи сообщения и машина состояния заданы файлами описания (скрипт-файлами, script). Интерфейс оператора интерпретирует скрипт-файлы состояния системы в соответствии с действиями оператора с целью отображения экранов и управления системой PCNC.

Рассмотрим процесс создания открытого языкового процессора на примере интерпретатора.

Концепция интерпретатора управляющей программы состоит в его построении по типу геометрического ISO-процессора [51]. Происхождение наименования связано с языком ISO-7bit управляющих программ, а суть концепции состоит в привлечении двух принципов организации программной среды для решения геометрической задачи. Согласно первому, ISO-процессор должен быть построен так, чтобы воспринимать операторы языка ISO-7bit управляющей программы как машинные инструкции. Второй принцип заключается в том, что реализация ISO-процессора осуществляется на основе объектно-ориентированного подхода. Концепция ISO-процессора представлена на рис. 124. В прямоугольных рамках обозначены возможности, вытекающие из указанных двух принципов.

Архитектура и система команд ISO-процессора. ISO-процессор представляет собой многозадачную систему, в которой работают четыре параллельных задачи: интерпретации, смежной коррекции, интерполяции и локальной диспетчеризации. Схема взаимодействия отдельных задач приведена на рис. 125.

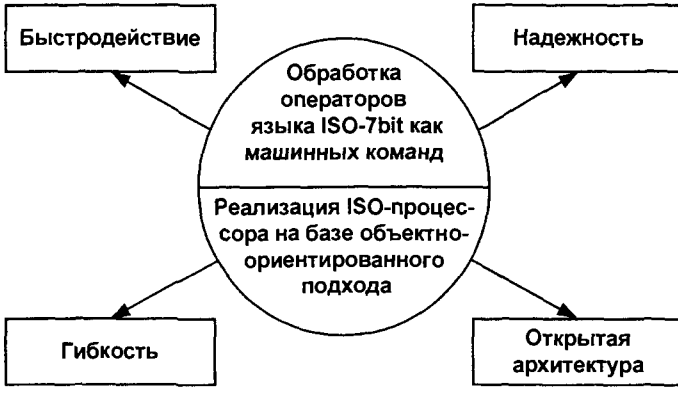


Рис. 124. Концепция ISO-процессора

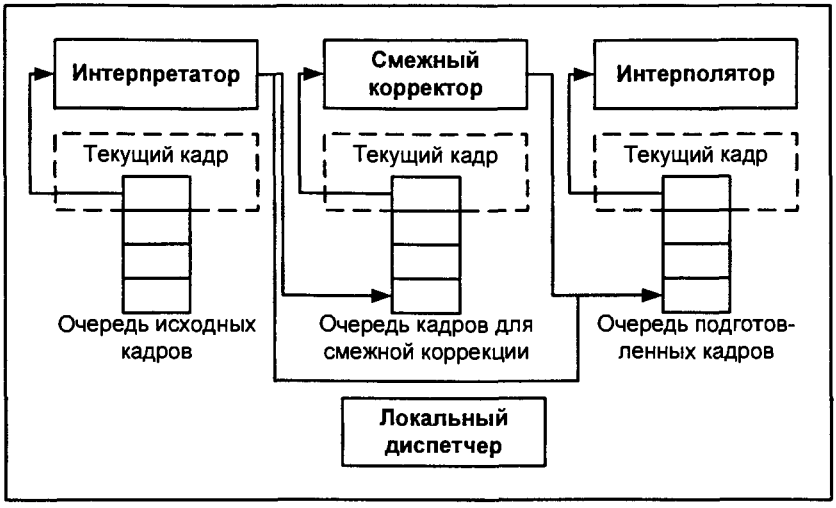


Рис. 125. Выделение параллельных задач в ISO-процессоре

Интерпретация состоит в подготовке кадров управляющей программы. Упорядоченные кадры УП пребывают в очереди исходных кадров. Интерпретатор обрабатывает первый кадр из очереди и переставляет его в очередь либо подготовленных кадров, либо кадров для смежной коррекции. Необходимость в смежной коррекции возникает при расчете эквидистантных контуров и заключается в коррекции существующих кадров или в синтезе дополнительных. Смежный корректор осуществляет коррекцию смежных кадров из своей очереди и переставляет их в очередь подготовленных кадров. При необходимости смежный корректор генерирует дополнительные кадры и вставляет их в ту же очередь подготовленных кадров. Если процесс смежной коррекции выключен, то очередь подготовленных для интерполяции кадров формируется непосредственно в процессе интерпретации. Функция интерполяции традиционна [76]. Необходимость диспетчеризации вытекает из взаимодействия задач в реальном времени и разделения между ними общих системных ресурсов.

Для формирования команд используем операторы языка ISO-7bit управляющей программы ЧПУ в ее традиционном виде, однако информацию управляющей программы трактуем иным образом. Кадр УП на языке ISO-7bit несет информацию о заявленных алгоритмах и структурах данных. Алгоритмы представлены подготовительными функциями (G-функциями). Структуру данных составляют функции размерных перемещений (X, Y, Z, I, J, K, R), подачи (F), скорости главного движения (S). Функции

структур данных можно рассматривать как параметры G-функций, а сами G-функции – как системы команд ISO-процессора.

Команды разбиты на группы: Plane – выбор координатной плоскости и переход к относительной системе координат; Dimension – преобразование размерности к форме, используемой в алгоритмах интерполяции; Correction – расчет эквидистант; Delay – задание выдержки времени; Interpolation – выбор алгоритмов интерполяции; StandartCycles – вызов стандартных циклов. В зависимости от версии языка ISO-7bit возможны и другие группы: Condition – организация перехода к следующему кадру; Acceleration – расчеты участков разгонов и торможений и др. В каждой группе выделены одна или несколько подгрупп ортогональных, т. е. взаимоисключающих, G-функций. Активные в данный момент подготовительные функции (команды) образуют G-вектор, размерность которого определяется количеством ортогональных подгрупп, а следовательно, зависит от конкретной версии языка.

Выделение групповых интерпретаторов. Структура системы команд предопределяет структуру процессора интерпретации. Интерпретация кадра осуществляется независимо для каждой группы. В разных версиях языка ISO-7bit обозначения подготовительных функций и их алгоритмическое наполнение не совпадают, и это вызывает определенные затруднения у разработчиков систем управления. Особенности предлагаемого подхода состоят в том, что сама структура системы команд отображается в ресурсах системы управления. Такая схема обеспечивает гибкость, поскольку функциональное назначение и возможности системы определяются множеством выбранных подготовительных функций. Конфигурация интерпретатора формируется структурой системы команд. Число групп G-функций задает количество групповых интерпретаторов, а в каждом интерпретаторе предусмотрены подгруппы, с которыми он работает, но не определены G-функции, которые входят в каждую подгруппу. Групповой интерпретатор обращается к подготовительной функции как к соответствующей координате G-вектора и передает ей управление. Использование подобной схемы означает, что одно и то же устройство ЧПУ может использовать различные системы команд. В этом состоят новые гипотетические возможности систем PCNC.

4.3.4. Стандартные средства поддержания открытой архитектуры

Операционная система Windows CE с гарантированным временем реакции 500 мкс превосходит в этом плане Windows NT, но в однокомпьютерных системах не удовлетворяет полностью требованиям реального времени. В этой связи неизбежно применение расширений реального времени. Время реакции, которое гарантирует такое расширение, на порядок меньше, чем у Windows CE. Одно из самых удачных решений сегодня –

это расширение RTX фирмы VentureCom, которое заменяет аппаратно-зависимый уровень HAL ядра Windows NT для обеспечения необходимой реакции на события. При этом появляются новые объекты ядра с идентичными для прежних объектов механизмами взаимодействия, что и позволяет рассматривать Windows NT и расширение RTX как одно целое, т.е. как операционную систему реального времени (ОСРВ).

Рассмотрим объекты и механизмы Windows NT, которые позволяют строить модули системы PCNC по типу открытого языкового процессора. Будем оперировать понятиями «процессы», «потоки», «таймеры», «сообщения». Процесс – это экземпляр (копия) выполняемого приложения. Процессы в 32-битной среде Windows (Win32) инертны, они ничего не выполняют; им отведено адресное пространство объемом 4 Гбайт, где содержатся код и данные выполняемого приложения. Для того чтобы Win32-процесс что-то выполнял, в нем создается поток (thread – нить). Один процесс может содержать несколько потоков, тогда они одновременно выполняют код в адресном пространстве процесса. Для этого каждый поток располагает собственным набором регистров процессора и собственным стеком.

В системе PCNC сосуществуют несколько процессов, которые обмениваются данными и сообщениями. Их делят на процессы реального времени и машинного времени (Win32-процессы). В процессах реального времени реализованы в виде отдельных потоков интерполятор, модуль Look-a-head (опережающего просмотра кадров), система поддержки коммуникационной среды. В свою очередь в процессе машинного времени работают потоки поддержки коммуникационной среды, терминальная задача (MMI), интерпретатор, запускающий групповые интерпретаторы. Упрощенная модель потоков системы PCNC представлена на рис. 126.

Рассмотрим проблему создания и синхронизации потоков в системе PCNC.

Создание потоков в системе PCNC. Чтобы определить, когда необходимо создавать потоки, исследуем проблему многопоточности. Многопоточность позволяет добиться минимального простоя процессора и более эффективной его работы. Например, прорисовка сложного экрана требует много времени, что недопустимо для систем реального времени. Выделение MMI в отдельный поток с низким приоритетом снимает эту проблему. При возникновении любой паузы ОС передаст MMI время для прорисовки экрана. Срабатывание таймера запустит высокоприоритетный поток интерполятора, и MMI-поток будет вытеснен.

В отдельные потоки выделяют любые длинные операции с файлами. Например, в редакторе УП в отдельные потоки вынесены операции: 3D-моделирования УП, расчета G-вектора в текущем кадре УП и т.д. Выделение модулей системы PCNC в отдельные потоки обеспечивает их независимость. Например, интерполятор, работающий в отдельном потоке,

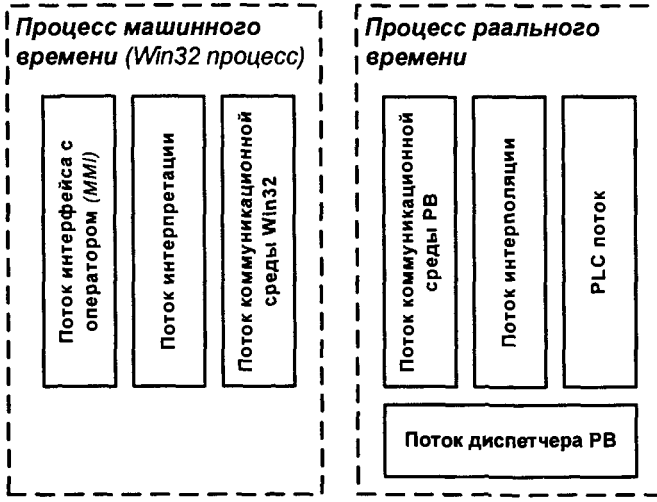


Рис. 126. Вариант потоковой модели системы PCNC

не зависит от фазы подготовки кадра интерпретатором, работающим в другом потоке. Непрерывность работы интерполятора обеспечивается буферизацией кадров между интерпретатором и интерполятором. Однако при выделении большого количества потоков возрастает время на их переключение. Особенно заметен этот эффект в быстрых процессах, таких как интерполяция.

Иногда следует избегать создания потоков. Потоки являются мощным инструментом, который следует применять осознанно. Нельзя располагать управляющие элементы (control-элементы) MMI-интерфейса в отдельных потоках, например ListBox (список), сортирующий и отображающий данные системы управления. В этом случае выделение потока породит Windows-проблему синхронизации при обработке сообщений в других потоках. Правильным решением здесь будет закрепление за всеми управляющими элементами отдельного потока, занимающегося вычислениями и сортировкой только в фоновом режиме.

Мы сталкивались с ситуацией, когда на одном из этапов проекта были вынуждены отказаться от многопоточности. Разрабатывали прототип MMI на базе многооконного интерфейса (Multiple Document Interface, MDI). Задача состояла в управлении в рамках одного процесса (MDI-приложения) несколькими системами ЧПУ, каждая из которых была закреплена за отдельным документом. Данные систем ЧПУ отображались в отдельных окнах MDIChild, работающих в своем потоке. Статусбар (StatusBar) и тулбар (ToolBar) были контекстно-зависимы, принадлежали родительскому окну MDIFrame и отображали состояние текущего окна MDIChild. При всяких

изменениях система ЧПУ уведомляла свое MDIChild окно. При этом обновлялись управляющие control-элементы и перенаправлялись изменения в поток родительского окна MDIFrame. Там в свою очередь обновлялись тулбар и статусбар.

Аналогично выполнялись транзакции в обратном направлении. Команды со стороны тулбара направлялись в поток активного MDIChild окна, которое обновлялось и отправляло по сети команды в систему ЧПУ.

Проблемы с вычислительными ресурсами начинались при одновременной работе более чем с тремя системами ЧПУ. Решение состоит в отказе от многопоточности и закреплении каждого экземпляра интерфейса оператора за отдельным процессом. В этом случае максимально используется вычислительная мощность компьютера. Переключение между отдельными экземплярами интерфейса оператора осуществлялось так же, как между приложениями.

Синхронизация потоков системы PCNC с помощью объектов ядра Windows NT. Использование многопоточности требует решения таких проблем, как работа с разделяемой памятью, синхронизация потоков, работа с разделяемыми ресурсами (например, портами ввода-вывода). Решения строятся на использовании стандартных объектов ОС (критические секции, мютексы, таймеры, семафоры, события), большая часть из которых является объектами ядра.

Формально потоки делятся на основные и порожденные от основных. Далее рассмотрены проблемы работы с основными потоками, поскольку порожденные могут использовать другие средства ОС для синхронизации потоков (блокировки потоков с помощью оператора Sleep(), таймеры Windows, передачи управления с помощью функции PeekMessage()).

Использование критической секции в разделяемой памяти рассмотрим на примере интерпретатора и интерполятора. Интерпретатор интерпретирует кадр и записывает его в формате IPD (Interpolator Data) в кольцевой буфер. Интерполятор считывает кадр из буфера и интерполирует его. На этапе записи и считывания кольцевой буфер должен быть заблокирован, чтобы избежать чтения неполного кадра или перезаписи неполностью прочитанного. Для этой цели создается критическая секция и инициализируется, например, в главном потоке процесса. В обоих потоках защищаемый код обрамлен вызовом функций EnterCriticalSection() и LeaveCriticalSection() в начале и конце. Если передача управления осуществлена в критической секции, второй поток не сможет войти в нее и передаст управление первому потоку для выхода из критической секции.

Другим объектом синхронизации потоков служит мютекс (mutex), который позволяет синхронизировать потоки разных процессов. Рассмотрим пример синхронизации данных коммуникационной среды между процес-

сами реального и машинного времени, передаваемых через разделяемую память. Мютекс-объект создается в коммуникационном потоке процесса реального времени с помощью функции CreateMutex(), которая сразу же возвращает описатель (handler) объекта. Win32-поток получает описатель мютекс-объекта с помощью функции CreateMutex() или OpenMutex(). Код в синхронизируемых потоках обрамлен функциями WaitForSingleObject() и ReleaseMutex() в начале и конце. Таким образом, блокируются попытки записи и считывания из разделяемой памяти соответственно коммуникационных потоков реального и машинного времени.

Семафор используют для управления ресурсами, например портами ввода-вывода в системе PCNC. Когда запрашивается ресурс, то ОС уменьшает содержимое счетчика ресурсов. Семафор создается с помощью функции CreateSemaphor(), в которой указано количество ресурсов, подлежащих мониторингу. Описатель семафора из другого потока создается с помощью той же функции или функции OpenSemaphor(). Синхронизация осуществляется в блоке, обрамленном функциями WaitForSingleObject() и ReleaseSemaphor(). В отличие от мютекса, освобождение которого возможно только занявшим его потоком, семафор может быть освобожден любым потоком.

Существует и другой объект ядра для синхронизации потоков – событие. В системах PCNC событие можно использовать, когда код инициализации системы или код выхода из ошибки выводится в отдельном потоке. Инициализирующий поток переводит объект-событие в состояние «занято» и приступает к своим операциям. В этот момент рабочий поток приостанавливает свое исполнение и ждет. По окончании инициализации поток инициализации возвращает объект-событие в состояние «свободно». Рабочий поток активизируется и продолжает свою работу. Использование события-объекта позволяет упростить диспетчеризацию процессов реального времени. Диспетчер построен на базе таймера. По сути, это отдельный поток с наивысшим приоритетом. При каждом срабатывании таймера вызывается его call-back функция, в которой реализована схема диспетчеризации. Согласно этой схеме управление потоками осуществляется путем изменения их приоритетов. На этом работа call-back функции таймера завершается. Останов и запуск процессов осуществляет сама ОС.

4.3.5. Использование стандартных инструментальных средств поддержания открытой архитектуры

По оценкам специалистов, создание продвинутой системы ЧПУ на базе одного компьютера занимает порядка двухсот человеко-лет. Возникают проблемы методики создания системы PCNC на стадиях проектирования, отладки и внедрения, поддержки проекта релизами, сопровождения эво-

люции проекта. Для построения открытой архитектуры объединяют стандартные инструментальные средства.

Опыт показал, что интеграция инструментальных средств возможна на основе технологии Microsoft-OLE-Automation. В качестве базы был использован Visual C++, который входит в состав наиболее мощного пакета средств разработки программного обеспечения под Windows-MSDN (Microsoft Developer Network) [77]. Организации, выпускающие инструментальное программное обеспечение, встраивают свои системы в MSDN, используя такие готовые компоненты пакета разработки, как текстовый редактор, система поддержки проекта Source Safe и т.д.

На рис. 127 показан проект системы PCNC в среде Visual C++, где каждый модуль оформлен в виде подпроекта. Подпроекты состоят в некоторой зависимости друг от друга, которая определяет очередность компиляции. Суть в том, что во время компиляции модуль PCNC должен уже иметь те скомпилированные модули, от которых он зависит.

Существует и основной проект (Master Project), компиляция которого обеспечивает перекомпиляцию всех остальных модулей. На рис. 128 в качестве примера приведена схема зависимости модулей объектно-ориентированной магистрали (OOC, object oriented channel), которая показана стрел-

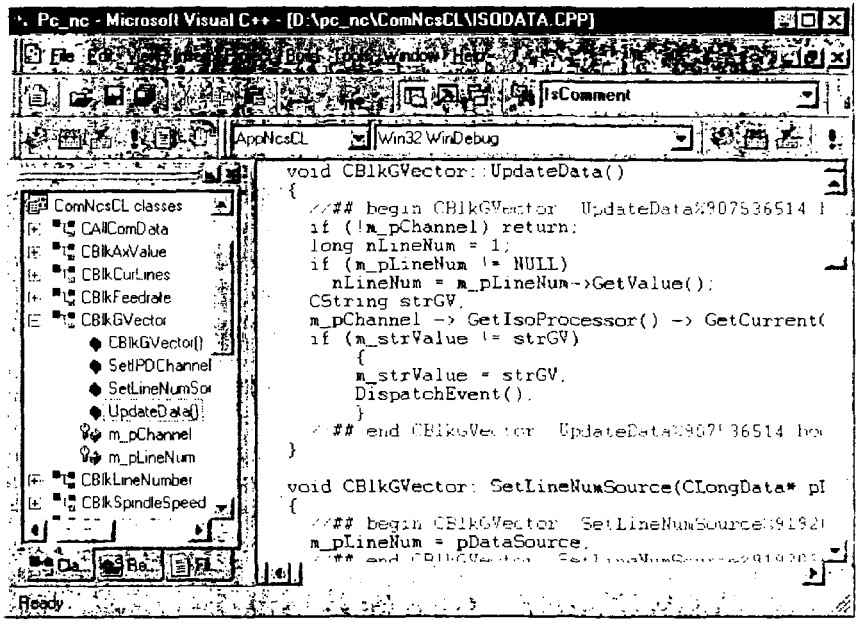


Рис. 127. Проект PCNC системы в среде Visual C++

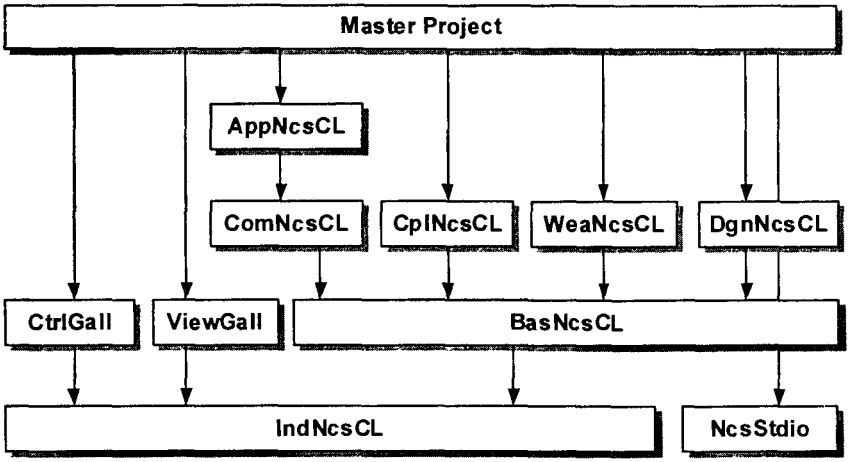


Рис. 128. Схема зависимости модулей объектно-ориентированной магистрали, определяющая очередность их компиляции

ками. Каждый модуль проекта оформлен в виде динамической библиотеки и имеет свое назначение:

- модуль, независимый от конкретной реализации системы PCNC (IndNcsCl);
- модуль работы с файловыми потоками (NcsStdio);
- галерея управляющих элементов (Ctrl Gall);
- галерея элементов визуализации (ViewGall);
- базовый модуль (BasNcsCL), в котором осуществляется открытие-закрытие объектно-ориентированной магистрали, а также ее начальная инициализация;
- коммуникационный модуль (ComNcsCL) геометрического канала;
- модуль языка УП высокого уровня (CplNcsCL);
- модуль диагностики программируемого контроллера (DgnNcsCL);
- модуль работы с сообщениями (WeaNcsCL);
- модуль диагностики следящих приводов по типу осциллографа (OscNcsCL);
- модуль прикладного уровня (AppNcsCL), поддерживающий реализацию приложений, взаимодействующих через объектно-ориентированный канал, и т.д.

Visual C++ обладает мощным механизмом настройки средств разработки, утилитами, обслуживающими проект, и механизмами встраивания внешних приложений. Системы поддержки проекта Source Safe и проектирования Rational Rose встраиваются в Visual C++ как внешние

приложения. Определим функции перечисленных систем и общую схему интеграции.

В проект создания системы PCNC вовлечены десятки человек, в него постоянно вносятся изменения и коррекции. Конечные пользователи системы нуждаются в поддержке релизов и обновлении версий, что определяет направление развития проекта. Поэтому проект нельзя начинать без системы поддержки проекта Source Safe (Microsoft). Как компонент MSDN она наилучшим образом интегрируется с Visual C++ и реализует простую концепцию работы с исходными кодами: в любой момент времени файл доступен для изменения только одному разработчику с помощью функций CheckOut и CheckIn. Во время работы с файлом другие разработчики, как правило, пользуются его копией. Пример проекта системы PCNC приведен на рис. 129. Использование Source Safe предполагает выделение администратора, который распределял бы права пользователей, маркировал релизы и архивировал версии, отслеживал корректность базы данных.

Архитектор проекта не в состоянии отследить взаимосвязь между используемыми классами и довести до программиста детали их реализации. Проблемы подобного рода решаются с помощью систем CASE. Одной из наиболее удачных среди них является Rational Rose [78]. Система позволя-

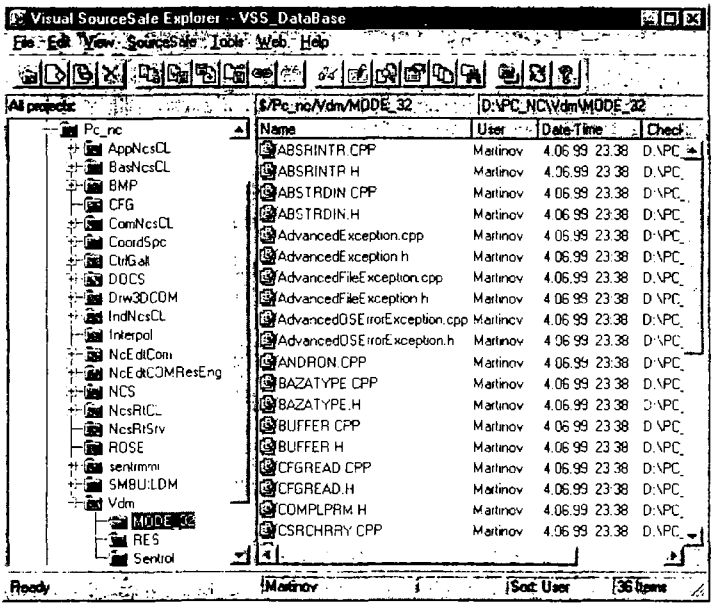


Рис. 129. Ведение проекта PCNC с помощью Source Safe

ет вести итеративное проектирование, осуществлять генерацию каркаса исходного кода, а также предусматривает интеграцию с Visual C++ и использование Source Safe.

Rational Rose поддерживает практически все распространенные нотации представления моделей, в том числе и UML (Unified Modeling Language) [71]. Фрагмент диаграммы классов в нотации Booch для модуля интерпретатора системы PCNC, разработанного в Rational Rose, представлен на рис. 130. Прикладные компоненты описываются системой Rational Rose с помощью четырех представлений объектно-ориентированного подхода [70] (рис. 131). Логическое представление задает ключевые абстракции и механизмы, формирующие предметную область и архитектуру системы PCNC. Физическое представление определяет конкретную программно-аппаратную платформу. Все это описывается диаграммами классов, объектов, модулей, процессов.

Если рассматривать систему со стороны статики-динамики, то здесь интерес представляют диаграмма переходов из одного состояния в другое и диаграмма взаимодействия и обмена сообщениями, определяющая порядок их передачи.

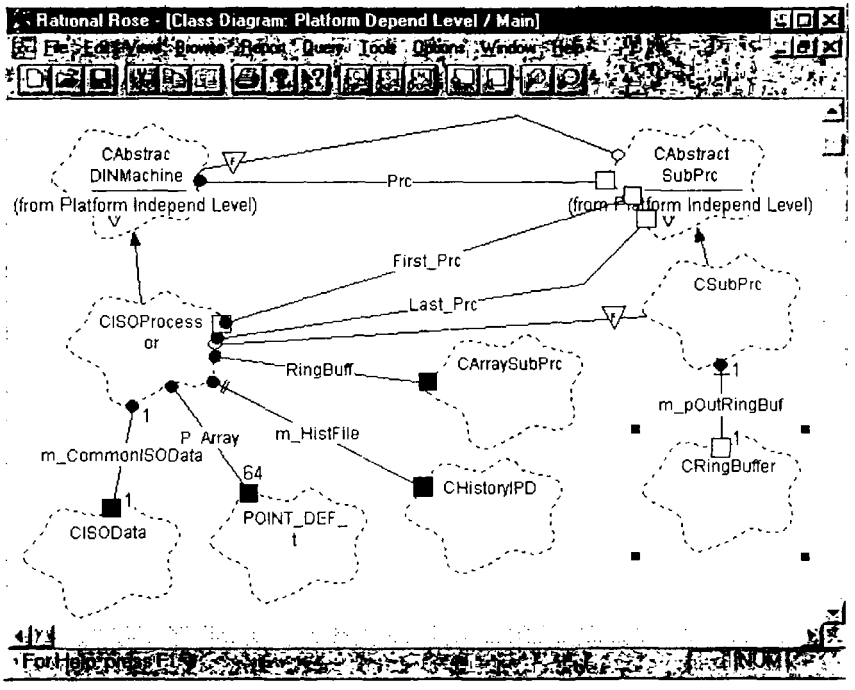


Рис. 130. Проектирование системы PCNC с помощью Rational Rose

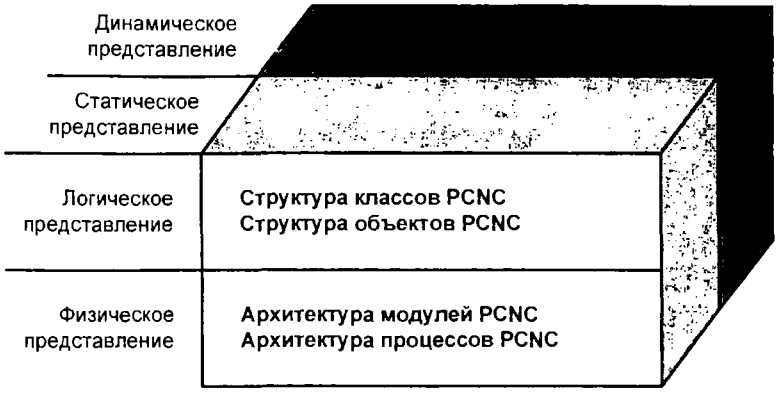


Рис. 131. Основные представления системы PCNC, поддерживаемые Rational Rose

4.3.6. Использование оригинальных инструментальных средств поддержания открытой архитектуры системы ЧПУ

Использование лишь стандартных средств ОС и инструментальных средств для разработки системы PCNC недостаточно. Опыт показывает, что ряд задач в системе PCNC поддаются формализации. При этом наиболее актуальны такие задачи, как создание каркаса модулей или внешних приложений, подключаемых к объектно-ориентированной магистрали, и создание каркаса машины состояния любого модуля или внешнего приложения системы PCNC. Под созданием каркаса программного модуля понимаем добавление в проект или подпроект тех классов и их объектов, которые реализуют базовые функциональные возможности, практически однотипные для подобных же программных модулей. Во всех случаях формализации процесса разработки системы PCNC необходимо создавать собственный инструментарий.

Создание скелета модуля системы PCNC с помощью мастера приложения NcsAppWizard. В общем случае AppWizard представляет собой средство создания и конфигурирования нового проекта в среде Microsoft Visual C++. NCsAppWizard, в частности определяет в интерактивной форме свойства и функциональные возможности будущего модуля системы PCNC. Скелет модуля создается на базе классов коммуникационной среды, при этом осуществляется генерация большей части необходимых программных кодов. Сведения о создаваемом приложении запрашиваются в диалоге с пользователем, а в результате генерируется исходный код. Таким образом, разработчик избавляется от одной из самых трудоемких работ по созданию скелета нового приложения.

Установка NcsAppWizard в систему разработки состоит в копировании файла с расширением «.AWX» в директорию Template Microsoft Visual C++. После этого NcsAppWizard становится доступным в качестве одного из мастеров создания приложений. Создание нового приложения при помощи NcsAppWizard представляет собой обработку последовательно сменяющихся диалогов. На первом шаге вводят информацию о типе приложения и целесообразности генерации примера. На втором шаге осуществляют ввод расширенных свойств приложения (например, для генерации обработчиков исключений), а также информацию для начального конфигурирования тулбара (количество кнопок и их расположение). На третьем шаге определяют стиль и обозначают права на приложения. Четвертый шаг задает размеры главного окна (при его наличии), возможности изменения размеров, автоматического центрирования и коррекции размеров окна при изменении разрешения экрана. Далее настраивают статусбар, параметры подключения к коммуникационной среде и т.д. В заключительном диалоге описывают те параметры проекта в текстовом виде, которые будут сгенерированы. В случае необходимости можно вернуться к предыдущим шагам и внести исправления.

В результате использования мастера разработки NCsAppWizard генерируется скелет модуля системы PCNC, в который добавлены классы и инициализированы их объекты для работы с общей объектно-ориентированной магистралью. На следующем этапе разработки осуществляется наполнение модуля функциональными возможностями. Частью этого этапа является разработка машины состояния.

Создание скелета машины состояния с помощью State Machine Builder. Под машиной состояния в системе управления понимают конечный автомат, реагирующий на управляющие воздействия и инициирующий необходимые действия [79]. Управляющими могут быть действия со стороны оператора (нажатие кнопки), события (например, поступление сигнала от датчика), возникновение ошибки. В результате управляющего воздействия машина состояния переходит в новое состояние, при этом выполняются функции, предусмотренные при переходе. Машину состояния описывают с помощью графа. Графы были использованы при разработке State Machine Builder (генератора машины состояния) – визуального инструментария разработки машины состояния. При этом были предложены новые понятия. Сложное состояние – логически обособленный фрагмент графа с иерархическими вложениями. Порты входа и выхода из сложного состояния – узлы, определяющие связь уровней сложного состояния. Эти понятия лежат в основе иерархических графовых структур, причем в вершине иерархии находится единственное сложное состояние. На каждой следующей ступени иерархии сложные состояния раскрываются в виде совокупности простых и сложных состояний.

Инструмент State Machine Builder позволяет не только визуально представить машину состояния в виде иерархического графа, но и задать имена классов, реализующих состояния и переходы, а также определить файлы их расположения. В результате инструментарий генерирует C++ код для машины состояния и встраивает его в Visual C++ проект системы PCNC. Инструментарий маркирует в генерируемом коде защищенные секции, в которых программисты реализуют функциональные наполнения переходов.

4.3.7. Формирование окружения разработки

Существуют два метода разработки программного обеспечения [80]. Метод «сверху вниз» характерен для процедурного подхода разработки. Метод итеративной разработки тех же фаз, но для последовательно детализируемых структур проекта наиболее удобен для объектно-ориентированного подхода (рис. 132). Итеративность процесса разработки PCNC обеспечивает ряд преимуществ. Она позволяет принимать во внимание изменение требований в процессе разработки, предъявляемых к отдельному модулю или системе в целом. Она предполагает постепенную интеграцию компонентов и модулей, которая может начинаться с меньшего количества интегрируемых элементов и позволяет свести риск разработки системы PCNC к минимуму, поскольку на этапе интеграции возможно обнаружение концептуальных ошибок.

Инвариантность создает возможность выпуска системы PCNC до общего завершения проекта с уменьшенными функциональными возможностями. Она также облегчает многократное использование компонентов путем анализа проектных решений на начальных итерациях, что позволяет

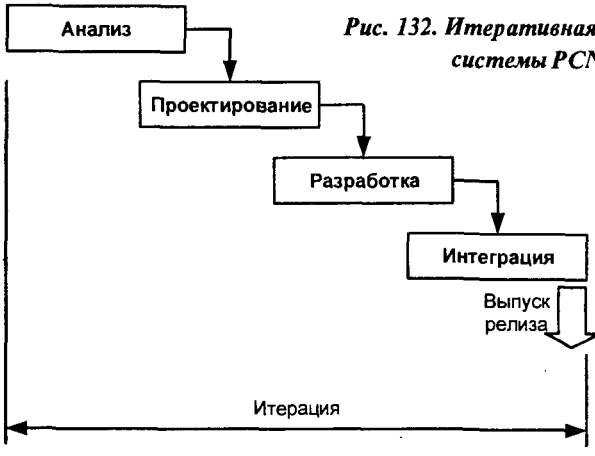


Рис. 132. Итеративная разработка системы PCNC

архитекторам сконцентрироваться на возможности повторного использования и совершенствования общего кода в последующих итерациях. А кроме того, приводит к более устойчивой архитектуре, поскольку слабые места обнаруживаются в ранних итерациях, когда критические параметры эффективности системы в целом легко обнаруживаются и исправляются.

Сопоставим стадии разработки, показанные на рис. 132, системе решений, рассмотренных ранее. Соответственно, Rational Rose и State Machine Builder закрепим за фазами анализа и проектирования, NCs AppWizard – за фазами проектирования и разработки, а Visual C++ и Win32 API – за фазой разработки. В результате получим целостную систему, наполняющую все фазы разработки и поддерживающую ведение проекта. Систему эту назовем «окружением разработки» открытой системы PCNC (рис. 133). В окружение разработки, помимо рассмотренных механизмов и инструментов, можно ввести дополнительный инструментарий, например систе-

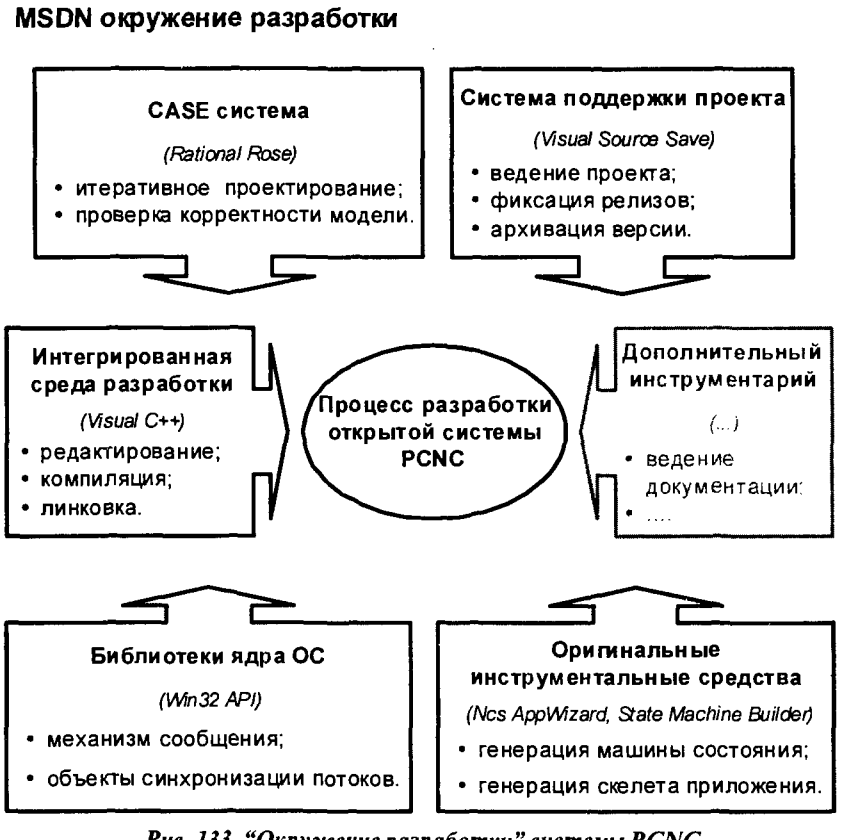


Рис. 133. "Окружение разработки" системы PCNC

му ведения документации (на рис. 133 показана серым цветом). Интеграция всего окружения осуществлена на базе MSDN.

Заключение

Открытость систем PCNC следует передать на уровень пользователей, которые являются их конечными потребителями. Построение модулей системы ЧПУ по типу открытого языкового процессора создает строгую архитектуру и конфигурируемую систему команд. Построение открытых языковых процессоров на базе объектно-ориентированного подхода решает проблему совместимости модулей. Ряд проблем разработки открытых систем PCNC может быть решен с помощью системных возможностей Windows NT, таких как многозадачность, синхронизация и обмен данными между задачами. При этом возникает возможность использовать отлаженные решения и накопленный опыт; что снижает затраты на разработку и повышает надежность системы. Стандартные инструментальные средства должны быть использованы для поддержания открытой архитектуры и формирования мощной подгруппы в окружении разработки системы PCNC.

Однако стандартных средств недостаточно, поэтому для формализованных задач необходимо иметь собственный инструментарий. Создание каркасов модуля (или внешнего приложения), подключенного к внешней магистрали, и машины состояния – задачи наиболее трудоемкие и в то же время формализуемые. Они могут быть решены соответственно мастером разработки Ncs AppWizard и генератором кода машины состояния State Machine Builder. Формирование окружения разработки позволяет предложить комплексное решение проблемы создания открытой системы ЧПУ, при этом конкретные решения закрепляются за отдельными фазами процесса разработки таким образом, что охваченным оказывается весь итерационный процесс разработки.

4.4. Технология компонентной организации программного обеспечения

Представлена компонентная организация программного обеспечения систем управления. Рассмотрены базовые понятия, изложены методические рекомендации по выбору компонентов и проанализированы сложные случаи создания компонентных моделей в системах управления, в том числе на базе стандартных библиотек MFC и ATL. Установлена приоритетная область использования COM-подхода в системах управления. Отмечена возможность инструментальной поддержки компонентного проектирования на основе формализма Г. Буча.

Высокие темпы эволюции технических требований к системам управления заставляют регулярно выпускать новые версии программного обеспечения. Это ставит перед производителями серьезные проблемы, в том числе и инвестиционного характера. Их решение видится на основе компонентной архитектуры, которая предполагает выделение компонентов, связываемых непосредственно в процессе работы системы управления («runtime»). Именно на эту возможность авторы стремились обратить внимание в этой статье.

Компонентная архитектура является развитием модульной реализации систем управления, при которой конкретная конфигурация собирается из готовых модулей. Однако компоненты привносят в программное обеспечение новые возможности. Так, компоненты можно подключать к приложению и отключать от него. Для этого они должны удовлетворять двум требованиям: компоноваться динамически и скрывать (инкапсулировать) детали внутренней организации. При этом компонентный подход использует все возможности объектно-ориентированного подхода.

Далее представлен необходимый понятийный аппарат и приведен пример, иллюстрирующий введенные понятия, приведена классификация COM-интерфейсов и COM-серверов, указаны области целесообразного использования компонентов, отмечена возможность инструментальной поддержки компонентного проектирования.

4.4.1. Базовые понятия

Компонентная модель COM лежит в основе таких технологий разработки прикладного программного обеспечения систем управления, как DCOM, OLE, OLE Автоматизация, ActiveX и OPC (рис. 134). Компонентная объектная модель COM делает стандартную структуру объекта регулярной, управляет его жизненным циклом и общением с другими объектами [81]. Другими словами, COM определяет правила структурирования объектов и их распределения в памяти, создания и уничтожения объектов, взаимодействия объектов между собой. Компонентная модель существует в рамках клиент-серверной архитектуры, когда клиент и сервер (компонент) связаны через COM-интерфейс. COM-интерфейс специфицирует функциональные возможности, которые компонент предлагает некоторой программе или другим компонентам.

От стандартного COM-интерфейса IUnknown произведены все остальные. Интерфейс IUnknown обязателен в любом компоненте и предназначен для (C++)-программистов; он обращает к методам (функциям) интерфейса через таблицу виртуальных методов [82]. Располагая указателем на IUnknown, клиент может запросить и другие интерфейсы компонента. Производный от IUnknown интерфейс наследует три базо-

Рис. 134. Технологии разработки прикладного программного обеспечения систем управления

вых метода: QueryInterface(), позволяющий клиенту получить указатель на любой интерфейс компонента из другого указателя интерфейса; Add() и Release(), поддерживающие механизм управления временем жизни клиента. С этой целью компонент хранит внутренний «счетчик ссылок» на своих клиентов. Если счетчик обнуляется, объект выгружает себя из памяти. При запросе указателя на интерфейс метод QueryInterface() неявно вызывает метод Add() для увеличения содержимого счетчика, а после окончания работы с интерфейсом клиент явно вызывает метод Release() для уменьшения содержимого счетчика ссылок.

Компонент может располагать одним или несколькими COM-интерфейсами; клиент, в свою очередь, может пользоваться сервисом от нескольких COM-интерфейсов. Это значит, что один COM-сервер может обслуживать нескольких клиентов, а один COM-клиент может получать сервис от нескольких компонентов (рис. 135).

COM-сервер и COM-интерфейс имеют свои глобальные уникальные идентификаторы GUID (Globally Unique Identifier). Один и тот же COM-сервер на двух разных компьютерах будет иметь один и тот же идентификатор GUID, а разные COM-серверы на разных компьютерах не могут иметь одинаковых интерфейсов. Уникальность GUID обеспечивается тем, что при изменении компонента или интерфейса утилита guidgen.exe генерирует

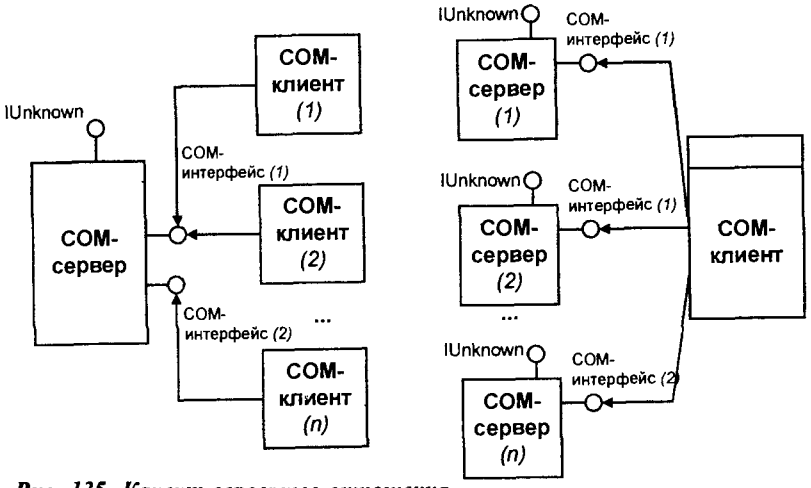


Рис. 135. Клиент-серверные отношения

новые идентификаторы на основе уникального сетевого адреса и точного времени запроса на создание GUID. Идентификаторы компонента (класса) и интерфейса обозначаются соответственно CLSID и IID.

Компоненты COM бинарно совместимы, что означает, что компонент будет совместимым для использования с любыми другими COM-компонентами независимо от языка, на котором он создан. Объекты и компоненты, разработанные на разных языках программирования и работающие в различных операционных системах, могут взаимодействовать без каких-либо изменений в двоичном (исполняемом) коде.

Любой клиент, желающий воспользоваться сервисом компонента, нуждается в предварительных сведениях о его интерфейсе. Существует технология, которая обеспечивает клиентов информацией о типах компонента. Файл, содержащий такую информацию, создается с помощью языка определения интерфейсов – IDL (Interface Definition Language).

Еще одно важное понятие – это фабрика классов (class factory). Фабрика порождает неинициализированный экземпляр объекта класса, т. е. некую заготовку, из которой впоследствии создаются экземпляры. Пусть приложение служит для управления автоматической линией, в которой работают десятки различных приводов с разными контроллерами. При разработке системы управления необходимо разработать компонент для каждого контроллера. Фабрика классов сокращает подобную работу, она генерирует полуфабрикат в виде экземпляра объекта класса, из которого затем инициализируются компоненты для каждого контроллера приводов. Фабрика классов сама представляет собой компонент со стандартным ин-

терфейсом IClassFactory2, способный создавать компонент с идентификатором CLSID; она не рассчитана на реализацию интерфейсов создаваемого компонента.

Компонентное программное обеспечение можно разрабатывать только на C++, но это необязательно. Для ускорения используют стандартные библиотеки MFC (Microsoft Foundation Classes) или библиотеки шаблонного программирования ATL (Active Template Library) [83].

4.4.2. Иллюстрация компонентного подхода на примере контроллера привода подачи

Рассмотрим следящий привод с обратными связями по току, скорости и положению. Контроллер привода осуществляет управление движением, принимает информацию о текущей координате, позволяет настроить параметры привода и считать эти параметры, запускает встроенный тест и т.д. Создадим COM-сервер, взаимодействующий с таким контроллером.

Объявление обобщенного интерфейса управления приводом с помощью библиотеки MFC выглядит следующим образом:

```
interface IDriveControl : public IUnknown
{
virtual HRESULT __stdcall Start()=0; // Запуск привода
virtual HRESULT __stdcall Stop()=0; // Останов привода
// Выход в заданную точку с заданной скоростью
virtual HRESULT __stdcall SetCommandPosition(VARIANT Position,
VARIANT Speedrate)=0;
// Получения текущей координаты
virtual HRESULT __stdcall GetActualPosition(VARIANT* pPosition,
VARIANT* pSpeedrate)=0;
// Конфигурация параметров
virtual HRESULT __stdcall SetConfigParameter(long Index,
VARIANT Value)=0;
virtual HRESULT __stdcall GetConfigParameter(long Index,
VARIANT* pValue)=0;
// Выполнение одного из внутренних тестов
virtual HRESULT __stdcall SelfTest(long Index, long* pResult)=0;
};
```

Все методы интерфейса должны возвращать значение типа HRESULT (т. е. дескриптор результата).

Интерфейсы COM-объектов (компонентов) предоставляют определенный сервис. Рассмотрим пример компонента с интерфейсом IDriveControl. Функциональные возможности интерфейса сосредоточены во вложенном классе XDriveControl.

```
class CDriveServer : public CCmdTarget
{
 public:
 // Реализация интерфейса
 class XDriveControl : public IDriveControl
 {
 virtual HRESULT __stdcall QueryInterface (REFIID iid, LPVOID* ppvObj);
 virtual ULONG __stdcall AddRef();
 virtual ULONG __stdcall Release();
 virtual HRESULT __stdcall Start(); // Запуск привода
 virtual HRESULT __stdcall Stop(); // Останов привода
 // Выход в заданную координату с заданной скоростью
 virtual HRESULT __stdcall SetCommandPosition(VARIANT Position,
 VARIANT Speedrate);
 // Получения текущего положения
 virtual HRESULT __stdcall GetActualPosition(VARIANT* pPosition,
 VARIANT* pSpeedrate);
 // Конфигурация параметров
 virtual HRESULT __stdcall SetConfigParameter(long Index,
 VARIANT Value);
 virtual HRESULT __stdcall GetConfigParameter(long Index,
 VARIANT* pValue);
 // Выполнение одного из внутренных тестов
 virtual HRESULT __stdcall SelfTest(long Index, long* pResult);
 }
 protected:
 XDriveControl: m_xDriveControl;
 DECLARE_INTERFACE_MAP ()
}
```

Макрос DECLARE_INTERFACE_MAP инициирует объявление таблицы идентификаторов интерфейсов класса. Создание самой таблицы интерфейсов продемонстрировано ниже:

```
BEGIN_INTERFACE_MAP(CDriveServer, CCmdTarget)
 INTERFACE_PART(CDriveServer, IID_IDriveControl, DriveControl)
END_INTERFACE_MAP()
```

Глобальные уникальные идентификаторы для COM-сервера и COM-интерфейса выглядят следующим образом:

```
// {D8F12B00-AE82-11d1-9396-B75855783964}
static const GUID DriveServer = { 0xd8f12b00, 0xae82, 0x11d1,
{ 0x93, 0x96, 0xb7, 0x58, 0x55, 0x78, 0x39, 0x64 } };
   // {D8F12B01-AE82-11d1-9396-B75855783964}
```

```
static const IID IID_IDriveControl = { 0xd8f12b01, 0xae82, 0x11d1,
{ 0x93, 0x96, 0xb7, 0x58, 0x55, 0x78, 0x39, 0x64 } };
```

Чтобы запустить COM-сервер в конкретной системе, его следует зарегистрировать, например с помощью утилиты regsvr32.exe. При регистрации глобальный уникальный идентификатор и путь к серверу записываются в системный реестр Windows. Клиент не знает, где находится COM-сервер; он просто обращается к операционной системе для получения указателя на интерфейс IUnknown для сервера с идентификатором GUID.

Покажем исходный код, который нужен клиенту для обращения к компоненту. Пример доступа к COM-интерфейсу со стороны клиента выглядит так:

```
//...
IDriveControl* pDriveControl = NULL;
// получение указателя на интерфейс IDriveControl
if ( pUnk->QueryInterface(IID_IDriveControl, (void**)&pDriveControl) ==
 NOERROR )
{
 long SelfTestResult;
 pDriveControl->SelfTest(0, *SelfTestResult);
 // освобождение указателя, полученного через QueryInterface
 pDriveControl->Release();
}
```

Указатель на компонент pUnk был получен заранее, например через вызов системной функции CoCreateInstance(). Посредством вызова IUnknown::QueryInterface() клиент получает в переменной pDriveControl указатель на интерфейс IDriveControl. По этому указателю вызывается метод SelfTest() для выполнения внутреннего теста привода, после чего использованный указатель освобождаеется вызовом Release().

4.4.3. Классификация COM-интерфейсов и COM-серверов

При построении COM-сервера у разработчика есть альтернативы. В этой связи остановимся на классификации COM-интерфейсов.

Интерфейсы, производные от IUnknown, обладают высоким быстродействием и интуитивно понятны C++ программисту (см. рассмотренный выше пример реализации интерфейса IDriveControl). Интерфейсы, производные от IDispatch, поддерживают OLE автоматизацию (технологию, позволяющую встраивать программные пакеты в другие приложения) и работу с языками сценариев, такими как Visual Basic Script, Java Script и т.д. Вызов метода посредством интерфейса IDispatch осуществляется через таблицу имен. По имени метода функция IDispatch::GetIDsOfNames() воз-

вращает из таблицы имен идентификатор метода (DISPID). По этому идентификатору функция IDispatch::Invoike() вызывает сам метод.

Набор функций, реализованных с помощью IDispatch::Invoike(), называют диспетчерским интерфейсом, или disp-интерфейсом (рис. 136). На рисунке слева представлена виртуальная таблица интерфейса IDispatch, а справа показан disp-интерфейс. Следует иметь в виду, что из-за особенности механизма вызова методов через таблицу имен интерфейсы на базе IDispatch работают существенно медленнее в сравнении с интерфейсами, производными от IUnknown.

Дуальные интерфейсы представляют собой комбинации IUnknown и IDispatch, обладающие всеми их свойствами. Как правило, реализация этого интерфейса требует больших затрат времени на разработку. Таблица виртуальных функций интерфейса IDriveControl, реализованного по типу дуального, представлена на рис. 137. По способам реализации COM-серверы подразделяются на следующие типы.

- Внутрипроцессный (in-process), загружаемый в ту же область памяти процесса, что и обслуживаемый клиент. Высокая скорость является основным достоинством внутрипроцессной связи. Клиенты получают доступ к функциональным возможностям внутрипроцессного сервера со скоростью вызова локальных функций, поскольку клиент и объект общаются напрямую через указатели интерфейса. Недостаток внутрипроцессного сервера состоит в его низкой устойчивости к ошибкам.
- Внепроцессный (out-of-process, local), когда клиент и сервер находятся на одном компьютере, но загружены в разные области его памяти (т.е. выполняются в разных процессах). Достоинство такой связи в высокой устойчивости к ошибкам. При ошибке в сервере, приводящей к завершению серверного процесса, клиент продолжит работу. Недостатком локального сервера можно посчитать низкую скорость: информация от од-

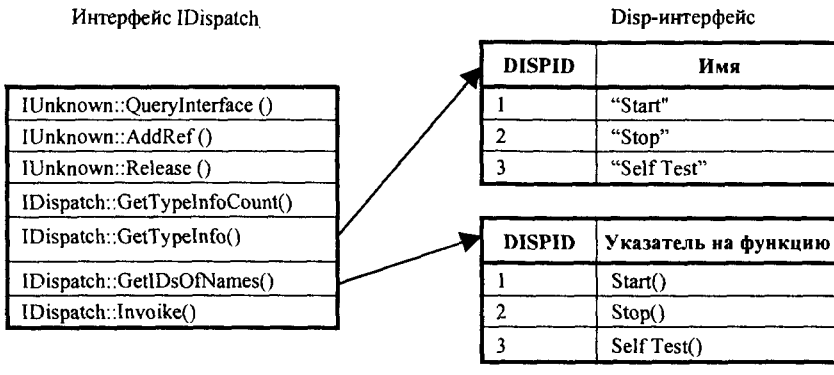


Рис. 136. Обобщенный интерфейс управления приводом на базе IDispatch

Таблица виртуальных функций объектов с интерфейсом IDriveControl
IUnknown::QueryInterface ()
IUnknown::AddRef ()
IUnknown::Release ()
IDriveControl::Start()
IDriveControl::Stop()
IDriveControl::SelfTest()

Таблица виртуальных функций объектов с дуальным интерфейсом IDriveControl
IUnknown::QueryInterface ()
IUnknown::AddRef ()
IUnknown::Release ()
IDispatch:: GetTypeInfoCount()
IDispatch::GetTypeInfo()
IDispatch::GetIDsOfNames()
IDispatch::Invoke()
IDriveControl::Start()
IDriveControl::Stop()
IDriveControl::SelfTest()

Рис. 137. Таблицы виртуальных функций объектов с интерфейсом IDriveControl

ного процесса к другому должна быть запакована, передана и распакована на границе процессов. COM берет этот сервис на себя.

Удаленный сервер расположен на другом компьютере по отношению к клиенту. Эта клиент-серверная связь наиболее медленная, поскольку здесь оказывают влияние пропускная способность и задержки в сети. Удаленная связь устанавливается с помощью протокола удаленного вызова процедур в распределенной модели COM (DCOM).

Объект COM может создавать и использовать другие COM-объекты, при этом клиенту не известно, является ли COM-сервер составным или монолитным. Использование одного компонента другим возможно посредством включения или агрегации.

Включение (containment) предполагает, что внешний компонент предоставляет интерфейс включаемому компоненту и обращается к нему для организации интерфейса. Создадим, например, компонент геометрического канала системы ЧПУ и вместо реализации функций управления приводом воспользуемся включенным компонентом управления приводом. Если клиент обращается к интерфейсу IDriveControl (рис. 138), компонент геометрического канала CMachineChannelServer переправляет вызов компоненту

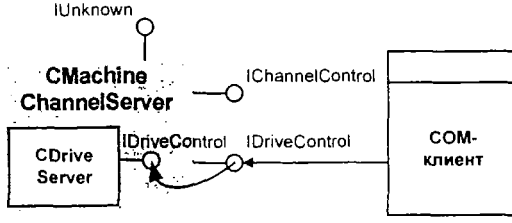


Рис. 138. Включение компонентов

CDriveServer. Внешний компонент может специализировать этот интерфейс, добавив свой код перед вызовом внутреннего компонента или после этого.

Агрегация (aggregation) означает, что внешний компонент агрегирует интерфейс внутреннего компонента, не создавая интерфейс заново и не передавая вызов этого интерфейса явно, как при включении. Вместо этого внешний компонент передает клиенту указатель на интерфейс внутреннего компонента.

Агрегация интерфейса IDriveControl компонентом геометрического канала показана на рис. 139. Она применяется тогда, когда реализация интерфейса устраивает разработчика полностью.

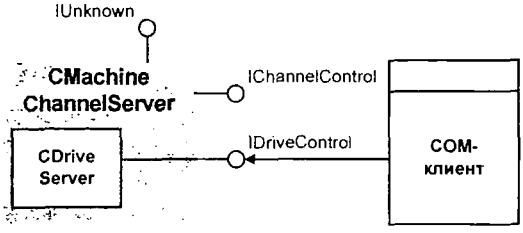


Рис. 139. Агрегация компонентов

4.4.4. Область использования COM

Преимущества компонентного подхода обеспечили широкую сферу его применения.

Во-первых, это повторное использование компонентов. COM позволяет однажды создать программный код, а потом использовать его во многих приложениях. Через какое-то время в компонент можно вносить коррекции и усовершенствования, что не повлечет необходимости менять любое использующее его приложение.

Во-вторых, это параллельная разработка. Обычно начинают с разработки интерфейсов компонента, что определяет корректность совместной ра-

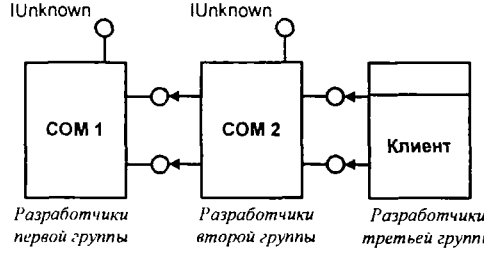


Рис. 140. Параллельная разработка программного обеспечения

боты всего программного обеспечения (рис. 140). Последующую разработку функциональных возможностей компонентов можно распараллеливать.

В-третьих, это унификация прикладного программного обеспечения. Речь идет о производстве, в котором собраны несовместимые системы управления разного типа и от разных производителей. Пусть нужно создать приложение, осуществляющее измерение сигналов для диагностики следящих приводов. Создают обобщенный COM-интерфейс и разрабатывают COM-сервер для каждого контроллера управления приводами. Прикладное приложение обращается через обобщенный COM-интерфейс к любому COM-серверу контроллера, при этом протокол управления конкретным контроллером привода полностью прозрачен (рис. 141).

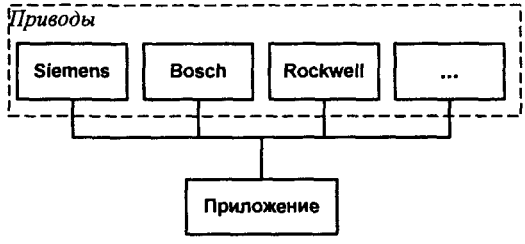


Рис. 141. Унификация программного обеспечения

Существуют некоторые особенности использования COM. Например, необходимо тщательно планировать интерфейсы, потому что опубликованный интерфейс нельзя менять. Если нужно изменить или расширить функциональные возможности интерфейса, то выпускают его новую версию с новым GUID. В новой версии компонента необходимо поддерживать все старые интерфейсы для обеспечения совместимости.

4.4.5. Инструментальная поддержка компонентного проектирования

Применение инструментальных средств существенно ускоряет процесс проектирования и разработки компонентного программного обеспечения. CASE-система Rational Rose 2001 ориентирована на разработку COM на базе библиотеки активных шаблонов ATL и использование только дуальных интерфейсов. Ниже приведено альтернативное решение проектирования COM-сервера для управления приводом с использованием стандартной нотации Буча [70].

Создадим абстрактный класс стандартного интерфейса IUnknown, который не будем генерировать (рис. 142). Объявим знакомые нам методы IUnknown как чистые виртуальные функции (pure function), т. е. не имеющие реализации.

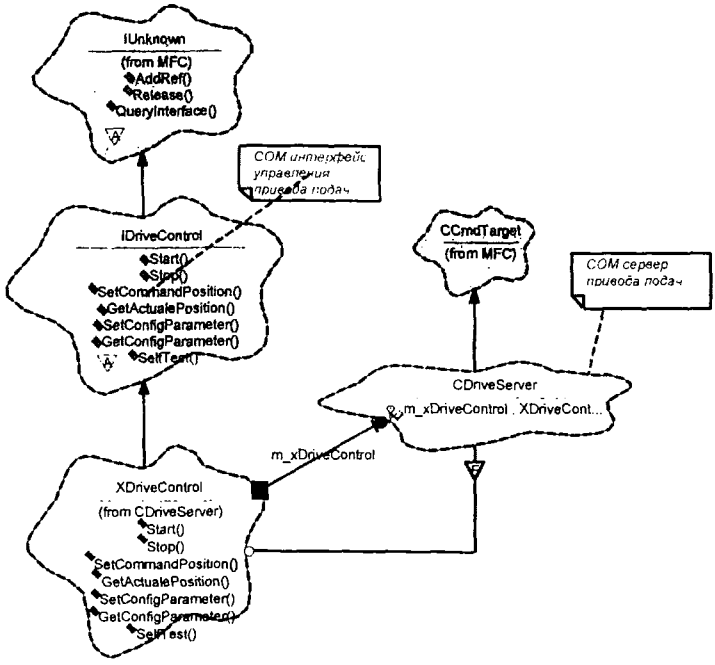


Рис. 142. Проектирование компонента в нотации Буча с помощью CASE-системы Rational Rose

Назначим расположение IUnknown в заголовочном файле предварительной компиляции. Создадим абстрактный класс интерфейса IDriveControl, унаследованный от IUnknown, в котором методы Start(), Stop(), SetCommandPosition(), GetActualPosition(), SetConfigParameter(), GetConfigParameter() и SelfTest() объявлены как чисто виртуальные функции.

Прототипом сервера служит класс CDriveServer, в котором объявлен вложенный класс XDriveControl, реализующий функциональность интерфейса IDriveControl. Построение интерфейса осуществляется путем переопределения в классе XDriveControl виртуальных функций Start(), Stop(), SetCommandPosition(), GetActualPosition(), SetConfigParameter(), GetConfigParameter() и SelfTest(). Сам класс компонента CDriveServer должен быть унаследован от стандартного MFC класса CCmdTarget или его потомка.

Сгенерировав C++ проект с помощью Rational Rose, получим каркас, который остается заполнить функциями компонента, реализующими функциональность COM-интерфейса.

4.4.6. Пример реализации ATL COM-сервера

Рассмотрим задачу верификации рабочего процесса в рабочем пространстве станка с ЧПУ с помощью твердотельного графического моделирования. Графический эмулятор работает совместно с ЧПУ-эмулятором в виртуальном времени, поскольку их скорости несопоставимы. Это означает, что можно запустить ЧПУ-эмулятор (включая процесс интерполяции) на определенное время, после чего он приостановит свою работу, пока не получит очередную порцию времени от графического эмулятора. Графические эмуляторы распространяются как коммерческие продукты. ЧПУ-эмуляторы разрабатываются производителями систем ЧПУ для своих целей. Проблема заключается в организации совместной работы эмуляторов, располагающих собственными интерфейсами. Для ее решения необходим модуль сопряжения на основе компонентного подхода.

В силу необходимости конвертировать форматы данных, перемещаемых между двумя приложениями (графическим и ЧПУ), предпочтительней использовать библиотеку шаблонов ATL, способную создавать компактный код.

Компонентная модель показана на рис. 143. ЧПУ-эмулятор предлагает прикладной интерфейс (API-функции) для взаимодействия с внешним окружением. ATL COM-сервер реализует интерфейсный модуль, управляющий, с одной стороны, ЧПУ-эмулятором через API-функции, с другой стороны, взаимодействующий с графическим эмулятором посредством COM-интерфейсов. Графический эмулятор через интерфейс Inc_emulator посредством ATL COM-сервера управляет ЧПУ-эмулятором. Интерфейс Inc_emulator позволяет выбирать управляющие программы в любом канале системы управления, запускать и приостанавливать выбранные программы в любом из каналов, а также полностью останавливать систему управления. Со своей стороны ATL COM-сервер через COM-интерфейс Igraphic_similator уведомляет графический эмулятор об изменении текущего кадра управля-

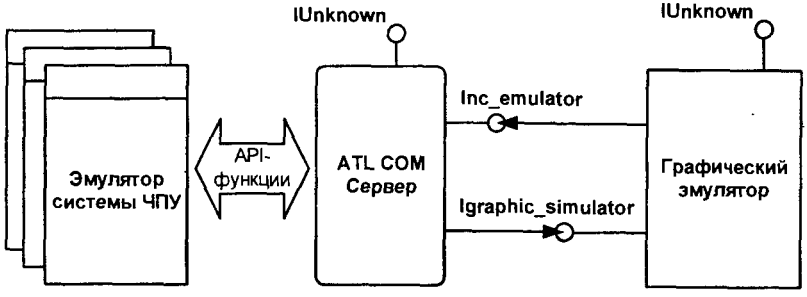


Рис. 143 Компонентная модель интеграции графического эмулятора

ющей программы, возникновении и снятии ошибки в системе ЧПУ, изменении конфигурации осей и каналов, о смене типа интерполяции в канале, вызове подпрограммы, а также об изменениях текущих координат осей.

COM-сервер поддерживает два потока асинхронного взаимодействия двух эмуляторов. Первый поток – основной, а второй предназначен для приема управляющих вызовов к ЧПУ-эмулятору и возврата управления. Однопоточная реализация COM-сервера позволила бы осуществлять лишь синхронные вызовы, а графический эмулятор вынужден был бы останавливаться и ждать ответа после каждого обращения к системе ЧПУ.

Заключение

Компонентная организация программного обеспечения систем управления позволяет повторно использовать исходный код, применять готовые компоненты независимых поставщиков, имеющиеся на рынке, компоновать систему управления под конкретные технологические задачи. Компонентный подход, применяемый на этапе проектирования программного обеспечения, дает возможность распараллелить процесс разработки за счет выделения компонентов.

Компонентная технология повышает надежность системы управления за счет повторного использования готовых отлаженных компонентов и сокращает время выпуска новых версий за счет возможности приобретения и интеграции существующих на рынке компонентов. Компонентный подход сегодня – это путь к крупномодульной реализации программного обеспечения систем числового программного управления мехатронными системами.

Глава 5.
Документы пользователя систем ЧПУ

К пользователям систем управления относят весьма обширную группу специалистов. Это технологи-программисты, наладчики систем управления, операторы. Всем им весьма полезны знания о внутреннем устройстве систем ЧПУ, но их преимущественно интересует информация о возможностях и тонкостях технологического программирования. К сожалению, технологи-программисты используют лишь небольшую часть таких возможностей, что, в конечном счете, оказывает огромное влияние на точность и производительность обработки.

Другая важная проблема состоит в том, что при разработке постпроцессоров систем автоматизированного программирования обычно предусматривают лишь слабое подмножество G-функций, а это приводит к недоиспользованию функциональных возможностей систем ЧПУ. Обозначенные выше специалисты заинтересованы и в других «документах пользователя», в числе которых рекомендации по конфигурированию системы управления, т. е. адаптации к конкретному объекту.

5.1. Структура руководства по программированию

Содержание документа по программированию систем управления в основном определяется описанием фазового пространства технологической машины, описанием кода ISO-7bit (общие принципы которого изложены в стандартах DIN66025 и ISO6983), указаниями относительно возможностей повышения языкового уровня кода ISO-7bit, комментариями к использованию подготовительных функций.

Код ISO-7bit до сих пор не потерял своего значения и непрерывно развивается за счет пополнения конкретных версий новыми подготовительными функциями. Набор этих функций служит хорошим отражением потребительских возможностей системы управления. Еще недавно классический диапазон G-функций составлял 100, и в этой связи версии кода ISO-7bit для различных систем

управления мало отличались одна от другой. Сегодня этот диапазон приближается к 1000. В этой связи поставлена задача рассмотреть те особенности кода ISO-7bit, которые обычно опускают при его поверхностном описании.

Любой инженер, связанный с ЧПУ, имеет более или менее полное представление о языке программирования систем с ЧПУ, коде ISO-7bit, принципы которого описаны в стандартах DIN66025 и ISO6983. Стандарты не накладывали ограничений на расширения языка, что оказалось весьма предусмотрительным. Резкое усложнение технологических возможностей мехатронного оборудования породило две тенденции в его программировании: код ISO-7bit существенно обогатился и сильно выросло число его версий, а кроме того, появились языки программирования более высокого уровня (например, язык CPL фирмы Bosch). Вторая тенденция имеет хорошую перспективу, но пока не стала доминирующей. Рассмотрим некоторые важные особенности расширенных версий кода ISO-7bit.

5.1.1. Фазовое пространство технологической машины

Под *фазовым пространством* будем понимать виртуальную параметрическую модель, одной частью которой является совокупность координатных систем машины, обрабатываемой детали, инструмента и отношений между координатными системами, а другой частью – скорость подачи, приведенной к инструменту, включая все виды коррекции подачи.

Совокупность координатных систем определяется некоторым множеством базовых точек, которые являются началом своих собственных координатных систем. Нулевая точка машины зафиксирована производителем технологической машины и привязана к поверхности и точке определенного ее узла. Координатная система с началом в нулевой точке машины является главной (исходной) для всех других координатных систем.

Нулевую точку заготовки задают относительно нулевой точки машины так, чтобы обработка заготовки была безопасной во всем рабочем пространстве машины. Нулевая точка программы служит точкой отсчета для всех данных управляющей программы. Ее выбирают с таким расчетом, чтобы не препятствовать обработке при последующей ориентации и зажиме заготовки, а также при работе измерительных циклов. Относительную точку устанавливают заранее для каждой координатной оси при помощи кулачка и конечного переключателя. Расстояние между нулевой точкой машины и относительной точкой (вдоль той или иной координатной оси) всегда постоянно.

После включения системы управления и до начала обработки обязательно должен быть выполнен выход во все относительные точки координатных осей для связывания нулевых точек технологической машины и

заготовки с помощью измерительных систем координатных приводов, а возможно и других измерительных устройств (датчиков касания заготовки, щупов, измерительных головок и др.).

Полученная информация после простейших перерасчетов сохраняется в памяти системы управления в виде смещения координатной системы управляющей программы относительно координатной системы технологической машины. Связывание координат осуществляют в самом начале управляющей программы вызовом подготовительной функции G54, которую называют в этом случае функцией программного смещения. Для организации нескольких разных программных смещений (например, при обработке по одной управляющей программе группы одинаковых деталей, закрепленных на общем столе технологической машины) используют несколько функций программного смещения G55,..., G59, значения которых сохраняются в памяти системы управления в специальной таблице. Отмену программного смещения осуществляют подготовительной функцией G53.

Некоторые системы управления допускают прямой ввод вектора программного смещения новой координатной системы управляющей программы относительно старой системы (с использованием подготовительной функции G92) или новой координатной системы управляющей программы относительно координатной системы технологической машины (с использованием подготовительной функции G93). Следует оговориться, что эти номера подготовительных функций в других системах управления могут быть использованы совершенно для других целей.

Для инструмента также определены базовые точки, служащие началом соответствующих координатных систем. Относительная точка инструмента расположена на том узле машины, который несет в себе инструмент, например на передней плоскости шпинделя. Нулевая точка инструмента находится в определенном месте инструментальной наладки (державки), причем при установке наладки в шпиндель нулевая и относительная точки инструмента обычно совмещаются. Координатные системы машины, управляющей программы и инструмента становятся связанными после ввода коррекции на длину инструмента (например, с использованием функции H, численное значение которой указывает на адрес в таблице коррекций длины инструмента). Теперь отработка управляющей программы возможна и достаточна лишь на основе ее собственных данных.

Связывание координатных систем позволяет использовать при программировании такие возможности, как зеркальное отображение, масштабирование, поворот координатной системы детали. Зеркальное отображение наиболее просто осуществить в системе координат управляющей программы. Модальная подготовительная функция G38 «включает» зеркальное отображение, причем в этом же кадре должны быть определенным образом

(далее выделено) показаны оси для зеркального отображения: N<номер кадра>G38 X-1 Y-1 LF.

В более сложной ситуации подготовительная функция G37 и соответствующие координаты задают «полюс» зеркального отображения (т. е. начало некоторой новой координатной системы в координатной системе управляющей программы). Далее работает подготовительная функция G38 точно так же, как это уже было показано, но для новой координатной системы. Подготовительная функция G39 отменяет зеркальное отображение.

Упомянутые выше подготовительные функции G38 и G39 используют и для масштабирования размеров детали, но в рамках соответствующего формата кадра. Так, функция G38 активизирует коэффициент масштабирования, величину которого указывают в том же кадре следующим образом: N<номер кадра> G38 X<величина коэффициента> Y< величина коэффициента> LF. Коэффициент масштабирования от 0 до 1 уменьшает размеры, а коэффициент, больший 1, приводит к их увеличению. Неодинаковые для разных осей коэффициенты масштабирования используют только при линейной интерполяции. Подготовительная функция G39 отменяет масштабирование.

Поворот координатной системы управляющей программы осуществляется все той же группой подготовительных функций G37, G38, G39. Кадр с функцией G37 задает координаты «полюса» поворота координатной системы управляющей программы. Кадр с функцией G38 устанавливает величину угла поворота, например R30 (отрицательный угол поворота), R45 (положительный угол поворота). Функция G39 отменяет поворот и устанавливает в «0» все запрограммированные значения.

Продолжая обзор модели фазового пространства технологической машины, рассмотрим некоторые расширенные возможности программирования и коррекции скорости подачи.

При программировании скорости подачи важно спрогнозировать эффект двух противоречивых факторов: быстродействия и качества переходных процессов координатных следящих приводов подачи. Проблему решают с помощью специально выделенного набора подготовительных функций.

Функция G07 задает в программе всем приводам подачи максимальное ускорение при разгонах и торможениях. Соответствующее значение хранится в памяти системы управления в области «машинных параметров». Функция G06 позволяет свободно назначать ускорения независимо для каждой координатной оси: G06 X1.0 Z2.1 LF.

В этом примере заданное значение ускорения для оси X равно 1 м/с^2, для оси Z – 2,1 м/с^2, а для оси Y – по умолчанию ускорение максимально и равно величине, сохраняемой в памяти системы управления в качестве параметра. Если в одном из следующих кадров встретится подготовительная

функция G206, то запрограммированные значения ускорений будут сохранены в памяти системы управления и смогут быть вновь вызваны функцией G06 (без параметров) после любой серии изменений. Функция G07 устанавливает максимальное ускорение, сохраняемое в памяти системы управления (например, 8 м/с^2).

Функции G08, G09 программируют включение-выключение разгонов и торможений. Так, функция G09 требует разгона до установленной в кадре скорости подачи и торможения до полной остановки в конце каждого кадра. Функция G08 вместе с функцией G00 разгоняет в начале кадра приводы до скорости быстрого (холостого) перемещения и тормозит приводы в конце кадра. Далее разгоны и торможения активизируются только при изменении величины подачи и обходе острых углов (с изменением направления движения вдоль той или иной оси).

Законы разгонов и торможений поддаются программированию (с помощью подготовительной функции G408) или не поддаются (при использовании подготовительной функции G09). Параметрами программирования служит идентификатор уравнения кривой разгона-торможения, а также число циклов интерполяции, в рамках которых процессы разгона и торможения начинаются и завершаются. Что касается уравнений, то возможны линейные разгоны-торможения, а также разгоны-торможения по закону SIN^2. Последний закон предпочтителен. Число интерполяционных циклов при линейных разгонах-торможениях может быть установлено в пределах 2–40. Число интерполяционных циклов для SIN^2-закона разгонов-торможений выбирают из ряда: 3, 4, 5, 10, 15, 20, 40. Приведем примеры форматов программирования законов разгонов-торможений: G408 без параметров (по умолчанию разгоны и торможения линейны и работают в двух интерполяционных циклах); G408 SIN3 LIN5 (выбран закон SIN^2 для трех интерполяционных циклов, LIN5 игнорируется в силу предпочтения закона SIN^2); G408 LIN5 (работают линейные разгоны-торможения всякий раз в пределах пяти интерполяционных циклов).

В результате описанных выше мероприятий контурная скорость подачи становится достаточно гладкой, однако не таковыми могут оказаться проекции вектора контурной скорости на координатные оси. Существуют специальные алгоритмы совместного (т. е. без траекторных искажений) сглаживания подачи для каждого из координатных приводов. Для этого опережающим образом просматривают несколько кадров управляющей программы и корректируют значения координатной подачи в местах резкого изменения этих значений. Алгоритм, выполняющий подобную работу, называют Look-Ahead (просмотр вперед). Он является частью ядра программного обеспечения системы управления, но может быть выключен, если управляющая программа подготовлена на рабочей станции соответ-

ствующим образом. Для управления алгоритмом Look-Ahead выбирают подготовительную функцию и формат ее использования, например (для одной из систем управления) G286 L<значение>, причем значение = 0 | 1, где 1 активизирует алгоритм, а 0 деактивирует его.

5.1.2. Повышение языкового уровня управляющих программ

Одна из возможностей повышения языкового уровня состоит в использовании макросов. Макрос может быть описан в любом месте управляющей программы, но не ранее его вызова. Формат описания макроса выглядит следующим образом:

имя макроса
<тело макроса>
# #

От подпрограммы макрос отличается тем, что непосредственно вставляется в ту инструкцию управляющей программы, которая его вызывает. Вызов макроса осуществляется, например, подготовительной функцией G14, причем он может быть сделан в теле программы неоднократно. Пример вызова: N<номер кадра> G14 N=»<имя макроса>» [P<число повторений>] LF.

Другая возможность повышения языкового уровня состоит в параметрическом программировании. Нередко встречаются похожие детали, отличающиеся лишь несколькими размерами. Параметрическое программирование позволяет в этих случаях избегать каждый раз разработки новых управляющих программ. Параметры являются переменными, над которыми можно выполнять четыре арифметических вычислительных действия, которым можно сопоставлять условия переходов и организации циклов.

В качестве параметра, например, используют Q0...Q255. Параметр может быть приравнен к выражению, а его значение может быть присвоено адресам управляющей программы. В одном кадре может быть сделано несколько таких присвоений. В теле программы можно расставлять метки и использовать их для переходов. Синтаксис метки выглядит так: [<имя метки>]. Синтаксис безусловного перехода на метку определен таким образом: GOTO «<имя метки>». Оператор условного перехода IF используют вместе с параметрами и параметрическими выражениями: IF Q_n <оператор сравнения> <параметр или выражение> GOTO «<имя метки>».

Параметрическое программирование удобно выполнять в рамках макроса.

Проблемы компенсации погрешностей ходового винта и влияния ортогональных осей не имеют прямого отношения к разработке управляющей программы, но относятся к ее важнейшему «окружению». Компенсация

погрешностей ходового винта необходима лишь в тех случаях, если в следящем приводе подачи использован датчик обратной связи по положению «непрямого измерения», т. е. такой, который соединен либо с винтом, либо с двигателем.

Значения компенсации погрешностей определяют в следующей последовательности. Общее перемещение, подлежащее коррекции, делят на равные отрезки, например величиной 10–20 мм. Тем самым формируют множество точек, для которых должна быть определена коррекция. Общее число таких точек обычно ограничено, но может достигать нескольких тысяч. Система ЧПУ последовательно перемещает рабочий орган станка в точки коррекции в положительном направлении, а величина компенсации коррекции измеряется с помощью внешнего измерительного устройства, например лазерного, в виде разницы между показаниями позиционного датчика следящего привода и фактического положения в рабочем пространстве станка (с точностью 0,1 мкм). После достижения последней точки процедуру повторяют для движения в отрицательном направлении.

Результаты аттестации вносят в ASCII файл с именем типа <name><axis>.tab. Файл имеет следующий внутренний формат:

<координата начальной точки> <величина шага> <комментарий>

<P0> <N0> < комментарий >

<P1> <N1> < комментарий >

....., где P и N – компенсации при движении в положительном и отрицательном направлениях соответственно.

Все файлы загружают в «пользовательскую» память EPROM, после чего они становятся активными.

Деформация узла или станины в направлении, перпендикулярном траектории некоторой оси, не вызывая искажения траектории этой оси, может оказывать влияние на другие ортогональные оси. В этом случае используют так называемую кросс-компенсацию.

Возьмем в качестве примера прогиб портала станка вдоль оси Z, который влияет на точность перемещений вдоль ортогональных осей X и Y. В принципе же для каждой оси допустимы две серии кросс-компенсаций, источником которых служат разные другие оси. Для формирования файла кросс-компенсаций выполняют действия, напоминающие те, которые производились при компенсации погрешностей ходового винта. Выбирают рабочий участок «зависимой» оси, делят его на отрезки, назначая точки измерения на их границах, осуществляют программное перемещение в точки измерения, определяют компенсацию как разность между положением по программе и измеренным (с помощью внешнего измерительного устройства). Результаты компенсации вносят в ASCII файл с именем типа

<compensation><compensated_axis><source_axis>.tab. Файл имеет следующий внутренний формат (он аналогичен приведенному выше):

<координата начальной точки> <величина шага> <комментарий>
<P0> <N0> < комментарий >

<P1> <N1> < комментарий >
....., где P и N – компенсации при движении в положительном и отрицательном направлениях соответственно.

Файл, как и другие подобные файлы, загружают в «пользовательскую» память EPROM, после чего он становится активным.

5.1.3. Функциональные возможности системы управления, отражаемые в версии управляющей программы

Функциональная мощность версии управляющей программы хорошо отображается набором ее подготовительных функций. Далее продемонстрируем в табличной форме (табл. 10) подобные наборы для весьма продвинутых систем ЧПУ Bosch Typ3osa (фирма BOSCH, Германия) и Andronic 2000 (фирма ANDRON, Германия). Выше, при изложении материала за основу была принята версия Bosch.

Сравнение двух версий оказалось возможным только в диапазоне подготовительных функций от G00 до G289. Вне этого диапазона подготовительные функции существуют только в версии Bosch (табл.11). Обе версии обладают исключительно высокой мощностью, а каждая из них имеет свои интересные особенности. Так, в версии Bosch предусмотрены широкий спектр программирования ускорений и коррекции подачи, большое число вариантов смещений систем координат, прямой доступ из программы к глобальным переменным – машинным параметрам, межканальная связь по данным, использование возможностей смешанного программирования в абсолютных и относительных координатах и др. В версии Andron обращают на себя внимание сложные измерительные циклы, а также раздел программирования лазерной обработки. Сравнение вариантов показывает, что несмотря на универсальный характер систем управления они в значительной степени ориентированы на своего базового заказчика.

Заключение

Код ISO-7bit до сих пор не потерял своего значения и непрерывно развивается за счет пополнения конкретных версий новыми подготовительными функциями. Набор этих функций служит отражением потребительских возможностей системы управления. Еще недавно классический диапазон G-функций составлял 100, и в этой связи версии кода ISO-7bit для различных систем управления мало отличались одна от другой. Сегодня

Таблица 10. Версии управляющей программы для продвинутых систем ЧПУ Bosch Typ3osa

G-функция	Значения для версии Bosch	Значения для версии Andron
G00 – G04	Традиционные функции интерполяции для обеих версий	
G05	Круговая (винтовая) интерполяция со входом в контур по касательной	
G06	Программирование ускорений при разгонах и торможениях независимо для каждой оси	
G07	Программирование максимального ускорения для всех осей при разгонах и торможениях	
G08	Использование разгонов и торможений только при изменении подачи или обходе углов	
G09	Использование разгонов и торможений до нуля в каждом кадре	
G10 – G13	То же, что и G00, G01, G02, G03 соответственно, но в полярных координатах	
G14	Программирование величины коэффициента усиления по скорости следящего привода	Вызов макроса по имени
G15	Отмена G14	
G16		Свободный выбор базовой плоскости (включение одной из главных осей X, Y, Z обязательно)
G17 – G19	Традиционные функции выбора базовой плоскости для обеих версий	
G20	Выбор плоскости круговой интерполяции по декартовым координатам полюса. Далее предполагается программирование в полярных координатах относительно полюса	
G22	Активизация таблиц базы данных для компенсации геометрических погрешностей и смещений нуля	Вызов, в рамках основной управляющей программы, другой, которая может быть выполнена несколько раз.
G30		Программирование сплайна
G32	Нарезание резьбы в режиме линейной интерполяции без компенсирующего патрона	
G34	Скругление угла для двух соседних прямолинейных участков (с допустимым отклонением под адресом E)	
G35	Выключение сглаживания угла	
G36	Выключение запрограммированного при скруглении угла отклонения, которое становится равным машинному параметру	

Продолжение табл. 10

G-функция	Значения для версии Bosch	Значения для версии Andron
G37	Программирование точки для зеркального отображения или поворота координат	
G38	Активизация зеркального отображения, поворота координат, масштабирования	
G39	Отмена зеркального отображения, поворота координат, масштабирования	
G40 – G42	Традиционные функции коррекцнн инструмента для обеих версий	
G43		Коррекция на радиус инструмента для участков линейной интерполяцин при подводе к соседним участкам с внешней стороны контура
G44		Коррекция на радиус инструмента для участков линей ной интерполяции при подводе к соседним участкам с внутренней стороны контура
G53 – G59	Традиционные функции смещения нуля для обеих версий	
G60	Смещение координатной системы программы	
G61	«Доработка» кадра до запрограммированной позиции	
G62	Отмена доработки кадра до запрограммированной позиции	
G63	Деактивация корректора скорости подачи	
G64	Поддержание постоянной подачи в точке контакта фрезы	
G65	Поддержание постоянной подачи для центра фрезы	
G66	Активизация корректора скорости подачи	
G67	Отмена смещения координатной системы программы	
G68	Сопряжение прямолинейных эквидистантных участков по дуге	
G69	Соединение эквидистант путем включения дополнительных кадров	
G70 – G71	Соответственно дюймовая и метрическая системы измереннн для обеих версий	
G72		Отмена зеркального отображения и масштабирования
G73	В отличие от G01 программируется линейная интерполяция с доработкой каждого кадра до запрограммированной позиции (независимо от G61 – G62)	Программирование зеркального отображения и масштабирования

Продолжение табл. 10

G-функция	Значения для версии Bosch	Значения для версии Andron
G74	Одновременный выход в относительную точку для указанных координат	
G75	Движение измерительного устройства триггерного типа до касания с заготовкой	
G76	Выход в фиксированную точку в координатной системе станка	
G77		Выполнение ранее объявленного цикла несколько раз в равноудаленных позициях вдоль дуги окружности
G78	Присвоение некоторой оси статуса сверлильной	
G79	Отмена присвоения оси статуса сверлильной	Ускоренный и безопасный выход к началу объявленного ранее цикла
G80 – G86	Традиционные сверлильные циклы для обеих версий	
G87 – G89		Фрезерные циклы
G90 – G91	Программирование соответственно в абсолютной и относительной системах координат для обеих версий	
G93	Программирование подачи через время обработки	Программирование сдвига нуля по отношению к некоторой фиксированной точке, каковой может быть нуль программы или точка, в которой ранее было объявлено начало координат
G94	Программирование подачи в мм/мин для обеих версий	
G95	Программирование подачи в мм/об	Программирование подачи через время обработки
G97	Прямое задание частоты вращения шпинделя	
G101 – G106		Программирование лазерной обработки с различными вариантами задания параметров рабочего режима
G105	Программирование новой нулевой точки для «квазинепрерывной» оси (оси с очень большим перемещением), от которой ведется новый отсчет координаты	
G110		Программирование набора входных сигналов для внешнего программируемого контроллера
G111		Программирование приема набора выходных сигналов внешнего программируемого контроллера
G112	Отмена снижения подачи до уровня, чтобы было возможно торможение (отмена учета тормозного пути)	

Продолжение табл. 10

G-функция	Значения для версии Bosch	Значения для версии Andron
G113	Снижение подачи до уровня, при котором возможно торможение (учет тормозного пути)	
G114	Слежение за измененнем направления подачи для учета зазора в кинематике	
G115	Отмена слежения за изменением направления подачи	
G134	Скругление угла аналогично G34, но с заданным радиусом скругления	
G138	Компенсация положения заготовки в системе координат станка	
G139	Выключение компенсации положения заготовки	
G145	Включение внешней компенсации инструмента	
G146	Выключение внешней компенсации инструмента	
G150	Использование типа позиционирования «бесконечных» осей, как это указано в машинных параметрах	
G151	Программирование типа «бесконечных» осей	
G153	Отмена первого (дополнительного аддитивного) смещения осей	
G154 – G159	Первое дополнительное аддитивное смещение нуля	
G160	Инициируемое извне (например, со стороны программируемого контроллера) смещение нуля	
G161	Точный выход в позицию при ускоренном перемещении	
G162	Отмена точного выхода в позицию	
G163	Доработка кадра до запрограммированной позиции при движении как со скоростью подачи, так и при ускоренном перемещении	
G164	Доработка кадра до запрограммированной позиции со снижением подачи до нуля. Контроль (через приводы) попадания осей в «окно точного позиционирования»	
G165	Доработка кадра до запрограммированной позиции со снижением подачи до нуля. Контроль (через приводы) попадания осей в окно «грубого позиционирования»	
G166	Доработка кадра до запрограммированной позиции со снижением подачи до нуля	

Продолжение табл. 10

G-функция	Значения для версии Bosch	Значения для версии Andron
G167	Отмена инициируемого извне (см. G160) смещения нуля	
G175	Цикл бесконтактного измерения с помощью измерительной системы привода и сигнала, подаваемого измерительной головке в точках измерения	
G177	Программирование максимального крутящего момента для оси (нормально это значение сохраняется в качестве машинного параметра)	
G181		Программирование измерительного цикла калибровки измерительной головки путем касания эталонного кольца на столе станка
G182		Программирование измерительного цикла: измерение расстояния
G183		Программирование измерительного цикла: определение значений для точек измерения на наклонной плоскости вдоль прямой линии
G184	Цикл резьбонарезания	Программирование измерительного цикла: определение радиуса вала относительно его центра в нескольких точках измерения по окружности
G185		Программирование измерительного цикла: определение радиуса отверстия относительно его центра в нескольких точках измерения по окружности
G186		Программирование измерительного цикла: определение координат точки X, Y, Z
G187		Программирование измерительного цикла: калибровка измерительной плиты в определением положенин на столе станка
G188		Программирование измерительного цикла: определение длины инструмент относительно его нулевой точкн
G189	Программирование в абсолютных координатах по отношению к активной нулевой точке	Программирование измерительного цикла: установление поломки инструмснта с помощью измерительной плиты
G190	Программирование в абсолютной системе координат с возможным выполнением инструкций относительного программирования	Задание центра окружности в абсолютных координатах
G191	Программирование в относительной системе координат с возможным выполнением инструкций абсолютного программирования	Программирование центра окружности относительно начальной точки дуги

Окончание табл. 10

G-функция	Значения для версии Bosch	Значения для версии Andron
G192	Программирование нижнего ограничения на частоту вращения шпинделя	
G194	Дискретное изменение подачи для достижения запрограммированной подачи в конце кадра (мягкое ускорение)	
G200	Линейная интерполяция на быстром ходу без торможения до нуля в конце кадра	
G206	Сохранение в памяти максимальных значений ускорений для всех осей (в отличие от значения по умолчанию); использование этих значений при программном вызове G06	
G245	Внешняя компенсация инструмента (другое значение в сравнении с G145, которое деактивируется)	
G253	Отмена второго (дополнительного аддитивного) смещения нуля	
G254 – G259	Второе дополнительное аддитивное смещение нуля	
G275	Цикл бесконтактного измерения с помощью измерительной системы привода и сигнала для измерительной головки. В отличие от G175 каждая точка измерения программируется заново	
G281		Указание максимальной относительной доли оси в общем перемещении по запрограммированному контуру (служит для назначения максимальных ускорений при разгонах)
G282		Выбор координатной системы либо заготовки, либо станка
G283		Программирование измерительного цикла: определение координат (X, Y, Z) точек поверхности свободной формы
G284		Программирование движения магазина смены инструмента
G285		Программирование касания заготовки измерительной головкой для определения соответствующего смещения нуля
G286		Активизация-деактивизация режима Look-Ahead
G289		Передача значения радиуса инструмента в систему инструментального менеджмента

Таблица 11. Подготовительные функции для Bosch

G-функция	Значения для версии Bosch
G292	Программирование верхнего ограничения на частоту вращения шпинделя (см. G192)
G301	Программирование осциллирующего движения для одной из осей при линейной интерполяции
G345	Внешняя компенсация инструмента (другое значение по сравнению с G145, G245, которые деактивируются)
G350	Программирование параметров осциллирующего движения
G352	Программирование нуля заготовки, расположенной в наклонной плоскости относительно координатной системы станка
G353	Отмена программирования наклонной плоскости
G354 – G359	Программирование таблицы параметров наклонной плоскости заготовки
G360	Третье инициализируемое извне смещение нуля (см. G260)
G408	Программирование формы разгонов и торможений в каждом кадре и их продолжительности в циклах интерполяции
G445	Внешняя компенсация инструмента (другое значение по сравнению с G145, G245, G345, которые деактивируются)
G500	Мониторинг коллизий при эквидистантной коррекции: программируется число кадров, вовлеченных в мониторинг
G520	Программирование перемещений для интерполируемых осей одного канала с помощью данных, поступающих из другого канала
G521	Возвращение к обычным операциям (отмена G520)
G522	Привлечение интерполируемых осей одного канала к программированию перемещений из другого канала
G523	Программирование скорости подачи для осей, перемещение которых программируется из другого канала
G524	Программирование ускорения для осей, перемещение которых программируется из другого канала
G532	Цикл резьбонарезания для нескольких шпинделей
G543	Включение мониторинга коллизий
G544	Выключение мониторинга коллизий
G545	Внешняя компенсация инструмента (другое значение по сравнению с G145 – G445, которые деактивируются)
G590	Активизация объединения осей, которые программируются вместе (с указанием master-оси и slave-оси)
G591	Деактивация объединения осей
G608	Сглаживание ускорений для каждой из интерполируемых осей, задаваемое числом интерполяционных циклов
G645, G745, G845	Внешняя компенсация инструмента (см. G145, G245, G345, G445, G545)
G900	Программирование идентификатора SERCOS-привода подачи непосредственно в программе (зависимого или независимого от производителя)

этот диапазон приближается к 1000. Содержание документа по программированию систем управления в основном определяется описанием фазового пространства технологической машины, указаниями относительно возможностей повышения языкового уровня кода ISO-7bit, комментариями к использованию подготовительных функций.

5.2. Конфигурация систем ЧПУ

Обобщены параметры конфигурации (часто называемые машинными параметрами) и показана важность выбора и систематизации этих параметров. Проиллюстрирована зависимость между параметрами конфигурации системы ЧПУ и ее функциоанльностью и качеством обработки.

Подавляющее число производителей систем ЧПУ выпускают всего одну базовую модель, которая постепенно эволюционирует, при этом смена базовых моделей осуществляется сравнительно редко. Успех модели на рынке зависит от способности ее адаптации к бесконечно разнообразным запросам станкостроителей и конечных пользователей. Возможность адаптации в свою очередь определяется открытой архитектурой систем ЧПУ [84], спектром подготовительных функций и набором параметров конфигурации. Проблема выбора параметров конфигурации (называемых часто машинными параметрами) не нашла своего отражения в литературе. В этой связи остановимся на этой проблеме подробнее. При ее рассмотрении был принят во внимание опыт фирмы BOSCH, которым в определенной степени воспользовались при создании отечественной системы ЧПУ на базе персонального компьютера [85].

5.2.1. Представление параметров конфигурации в системе ЧПУ

Параметры конфигурации являются, по сути, глобальными переменными системы ЧПУ. Значения параметров устанавливают на уровне стыковки с технологическим оборудованием, после чего доступом к изменению значений располагают лишь лица, обладающие на то специальным правом. Технологу-программисту (который входит в группу конечных пользователей) предоставлена возможность временного изменения значений некоторых параметров в рамках управляющей программы с помощью специально ориентированных на то подготовительных функций.

Каждый параметр конфигурации имеет свой идентификатор, представленный девятиразрядным десятичным числом. Четыре старших разряда идентификатора служат номером группы, к которой приписан параметр, а два старших разряда номера группы являются кодом функциональной об-

ласти. Таким образом, возникает упорядоченная картина параметров конфигурации, которая показана в табл. 12. В таблице не приведены полные идентификаторы всех параметров, но указано общее их число в каждой группе. Всего в таблице упомянуты 260 параметров конфигурации.

Каждый параметр является сложной однотипной структурой данных, представление о которой дает рис. 144. Здесь показан пример описания параметра с идентификатором 100300004, который устанавливает тип движения для каждой координатной оси системы координат рабочего пространства станка.

Приведем комментарий к табличной структуре данных. Диапазон значений параметра указывает наименьшее и наибольшее его значения. Дискретность параметра означает минимально возможное его приращение в пределах диапазона значений. Обычно приращения заданы в десятичной форме, однако существуют и специальные формы задания. Так, форма 2x означает, что приращения могут составить 1, 2, 4, 8...; форма 10x – что приращения могут быть 1, 10, 100, 1000...; форма a2x – что приращения могут составить a, 2a, 4a, 8a...

Если форма не задана, то в диапазоне значений приращения могут иметь любое значение. Размерность относится к физической сущности параметра. Число элементов означает структуру параметра, т. е. некоторый набор его индивидуальных значений. Так, для приведенного на рис. 1 примера, число элементов соответствует возможному числу координатных осей.

Длина элемента равна объему памяти, выделенному для каждого индивидуального значения параметра в элементе, причем длина элемента соот-

Таблица 12. Упорядоченная картина параметров конфигурации

Функциональная область		Группа параметров		Нанменованне группы
Код	**Наименование**	**Код группы**	**Число параметров в группе**	
10	Координатные оси и шпинделн	1001	1	Параметр привода
		1003	19	Параметры координатных осей
		1005	8	Параметры скорости осей
		1010	4	Динамика осей
		1015	4	Позиционирование и точность осей
		1020	4	Программные ограничители осей
		1040	38	Параметры шпинделей
		1050	9	Параметры SERCOS-интерфейса
20	Программируемый контроллер, интерфейс	2010	4	Мониторинг позиционирования
		2060	19	Параметры программ контроллера
30	Программирование систем ЧПУ	3010	9	Вспомогательные функции
		3080	7	Параметры управляющей программы
		3090	7	Определения циклов

Окончание табл. 12

Функциональная область		Группа параметров		Наименование группы
Код	Наименование	Код группы	Число параметров в группе	
40	Коммуникация	4055	15	Настройка параметров периферии
		4075	8	Параметры входов-выходов
		4080	7	Настройка контроллера PROFIBUS
		4085	5	Настройка PROFIBUS-сети
		4086	8	Параметры DNC-интерфейса
50	Приложения	5090	2	Параметры приложения
60	Интерфейс оператора	6001	2	Страница MMI после включения
		6005	3	Специфика окон канала
		6010	3	Параметры, выбираемые в диалоге
		6020	6	Установка формата дисплея
70	Параметры канала	7010	4	Оси канала
		7020	1	Шпиндели канала
		7030	5	Коррекции подачи, подача, ускорения
		7040	3	Размерность и масштабирование
		7050	17	Параметры G-функций, допуски
		7060	8	Программирование и настройка
		7070	2	Параметры CPL-программ
80	Специфические функции ЧПУ	8001	4	Параметры процесса штамповки
		8002	9	Параметры области управления
		8003	1	Параметр точного программирования
		8004	1	Программирование времени обработки
		8005	1	Программирование в полярных координатах
90	Управление системой	9010	5	Пароли
		9020	2	Размерность компенсаций и масштабирование сигналов контроллера
		9030	2	Системные параметры интерполяции
		9040	2	Число каналов и выделение процессорного времени
		9050	1	Системный параметр

ветствует его типу. Один или два байта выделяются для элементов типов (integer unsigned), (integer signed), (string); четыре байта – для элементов типов (integer unsigned), (integer signed), (string), (real, floating decimal point), (real, with exponent); восемь байтов – для элементов типов (string), (real, floating decimal point), (real, with exponent). Число цифр включает целую и дробную части. Дробная часть означает число цифр после запятой. Формат представляет собой дополнение к типу данных. Доступ раздельно указывает права станкостроителя, настройщика инструментов, пользователя: w – право читать и записывать значение параметра; r – право читать значение параметра; прочерк означает отсутствие всяких прав. Примечание описывает класс параметра, который может быть общим, а может указывать

на принадлежность к определенному каналу ЧПУ, функциональной подобласти шпинделя, функциональной подобласти координатной оси. Преселекция элементов приводит их индексы и преселективные значения, имеющие разнообразный смысл.

100300004	Тип движения координатной оси

Для каждой оси установлено:
является ли она линейной или круговой;
ведется ли для оси подсчет "модуля перемещения".

Диапазон	Дискретность	Размерность	Число элементов	Тип данных	Длина элемента	Число цифр	Дробная часть	Формат	Доступ	Примечание
0...4	1	-	16	integer	1 byte	1	-	unsigned	w, r, r	-

Преселекция элементов, [x] – индекс							
[1] 1 [9] 0	[2] 1 [10] 0	[3] 1 [11] 0	[4] 2 [12] 0	[5] 1 [13] 0	[6] 1 [14] 0	[7] 2 [15] 0	[8] 0 [16] 0

Значения параметра в диапазоне значений:
0 – координатная ось в параметре 100100001 не была определена.
1 – линейная координатная ось. Перемещения ограничены установленным диапазоном и программируются в миллиметрах или дюймах.
2 – круговая координатная ось (бесконечная ось) с подсчетом модуля перемещения. Это специальный тип круговой оси без ограничений на величину перемещения. Для бесконечных осей допустимо любое число оборотов в одном и том же направлении. Но как только перемещение превышает величину модуля перемещения (называемого также модуль-фактором), текущая координата перерассчитывается и попадает в диапазон от нуля до модуля перемещения.
3 – круговая координатная ось. Перемещения ограничены установленным диапазоном и программируются в градусах.
4 – линейная координатная ось с подсчетом модуля перемещения, специальный тип линейной оси. Если запрограммированная абсолютная координата превышает модуль перемещения, система ЧПУ выдает сообщение об ошибке (в процессе отработки управляющей программы). Как только перемещение превышает величину модуля перемещения, текущая координата перерассчитывается и попадает в диапазон от нуля до модуля перемещения. Момент достижения величины модуля перемещения определяется следящим приводом автоматически при достижении программно-установленного (с помощью подготовительной функции G105) нуля координаты.

Рис. 144. Формат параметров конфигурации на примере параметра установки типа движения

Семантика параметров. Рассмотрим семантику параметров в наиболее важной их группе, принадлежащей функциональной области «Координатные оси и шпиндели».

В *группе 1001* приведены число используемых координатных приводов, признаки круговых и линейных осей и признак привода шпинделя. Шпиндельная ось может иметь свой собственный признак, позволяющий присваивать ей адрес «С», например при резьбонарезании.

В *группе 1003* каждой оси присваивается независимое от канала уникальное имя (адрес) и придается параметр, обозначающий, является ли ось «обрабатывающей» или вспомогательной, а также параметр, указывающий, является ли она бесконечной (см. рис. 1). Для бесконечных осей заданы признаки направления вращения в зависимости от предыдущего положения (по часовой стрелке или в противоположном направлении), а также в кратчайшем направлении. Оси имеют признак активности по умолчанию (все вместе или выборочно). Для каждой из них предусмотрена возможность сократить допустимый крутящий момент (в процентах от максимального) и указывается, может ли она быть использована в совместной работе с измерительным датчиком триггерного типа (касание и остановка).

Оси могут иметь признак «Hirth axes», разрешающий перемещение только по узлам заранее установленной решетки. Для совместно работающих осей (ведущая – ведомая) возможно включение мониторинга допустимого рассогласования. Может быть сформирована группа совместно работающих осей с указанием ведущей оси в группе. Для каждой оси предусматривается возможность статического (в покое) мониторинга максимально допустимого крутящего момента. Для вспомогательных осей предусмотрено разрешение или запрещение возможности процентной коррекции скорости (с панели оператора).

В *группе 1005* для каждой координатной оси указаны максимально допустимая рабочая скорость, скорость быстрого перемещения, скорость в ручном режиме немерных перемещений (медленная, средняя, высокая, быстрая) и мерных перемещений.

В *группе 1010* для каждой координатной оси обозначены максимально допустимое ускорение (замедление) в рабочем и ручном режимах, максимальный скачок скорости между соседними интерполяционными циклами, предел изменения ускорения между соседними интерполяционными циклами, связанного с изменением скорости подачи.

В *группе 1015* для каждой координатной оси указаны значение дискреты и число дискрет перемещения в ручном толчковом режиме по каждой оси. Предусмотрена возможность мониторинга установленного для оси расстояния до конечной точки в кадре программы и определено «грубое окно» попадания в запрограммированную точку.

В *группе 1020* обозначено положение одного или двух программно реализованных конечных переключателей для каждой координатной оси (в положительном и отрицательном направлениях).

В *группе 1040* указаны параметры шпиндельной группы: число шпинделей и признак их активности, признак аналогового или SERCOS-привода шпинделя, признак принадлежности шпинделя к соответствующей группе. Для шпинделя могут быть указаны признаки «внешних» шпинделей, управляемых программируемым контроллером, число зубчатых колес кинематической цепи в шпиндельной бабке и частота вращения каждого зубчатого колеса (минимальная и максимальная). Задана таблица частот вращения шпинделя соответственно условным командам вспомогательных функций. Установлено «окно частот вращения», при вхождении в которое формируется сигнал подтверждения. Определены максимальное ускорение шпинделя в обычном и позиционном режимах и шаги процентной коррекции частоты вращения шпинделя. Представлен синтаксис вспомогательных М-функций, имеющих отношение к управлению шпинделем.

В *группе 1050* указаны параметры SERCOS-приводов: производитель, признак присутствия привода в кольцевой оптоволоконной сети и номер кольца, SERCOS-адреса каждого привода, время доставки управляющей телеграммы каждому приводу, тайм-аут при запуске приводов, условная мощность оптической передачи сигнала, скорость передачи сигнала в оптоволоконном кольце в бодах.

Редактор параметров. Параметры конфигурации сохраняются в базе данных системы ЧПУ, а доступ к ним осуществляется с помощью редактора в специальном режиме системы ЧПУ. Редактор предназначен для:

- визуализации и редактирования параметров конфигурации или отдельных полей параметров;
- ввода параметров с некоторого периферийного устройства и вывода на периферийное устройство;
- генерации (и спецификации) новых параметров конфигурации в рамках существующих групп;
- создания новых групп параметров конфигурации.

Редактор поддерживает различные права доступа к параметрам.Параметры конфигурации имеют некоторые значения по умолчанию. Как правило, в конкретной ситуации изменению подлежит лишь некоторое их множество. Редактор позволяет сформировать это множество для более удобной работы с редактируемыми параметрами.

Заключение

Представленная информация свидетельствует о чрезвычайной важности как отбора параметров конфигурации, так и регулярного (стандартно-

го) их представления. Общий спектр параметров конфигурации характеризует функциональные возможности системы ЧПУ и степень ее гибкости. Доступ к редактированию параметров осуществляется в рамках специального режима и предоставляется дифференцированно и только персоналу очень высокой квалификации. Доступ к отдельным параметрам возможен из управляющей программы с помощью специально выделенных подготовительных G-функций.

5.3. Методика программирования станков с ЧПУ

5.3.1. Базовые понятия

Кадры программы. Система ЧПУ исполняет кадры программы последовательно, один за другим. Каждый кадр состоит из некоторой совокупности слов, которые в свою очередь содержат адресную часть и цепочку цифр. К примеру, кадр может состоять из девяти слов с адресами N_G_ X_Y_Z_F_S_T_M. Последовательность полноформатных слов выглядит, например, так: G00 X-23450 Y40 M03 S250. Незначащие нули цифровой части слова пропускают. Числа типа real записывают с десятичной точкой, причем незначащие нули в дробной части также опускают. Например, X100.500 соответствует X100.5. Число слов в кадре переменно. Слова, описывающие перемещения, могут иметь знак (+/–). При отсутствии знака перемещение полагается положительным.

Модальный эффект. Большинство слов модальны. Это означает, что они остаются в силе на протяжении нескольких кадров, пока значение слова не изменится или функция, представленная словом, не будет выключена. Пусть, например, с помощью функции G1 запрограммирована линейная интерполяция с некоторой скоростью подачи. В последующих кадрах эта функция сохранит свою активность, пока интерполяция не изменится на круговую (функция G2) или линейную с ускоренной подачей (функция G0). Слова, которые действуют только в своем кадре, – немодальны.

Слова имеют смысл инструкций *(например, при задании типа перемещений вдоль координатных осей X, Y, Z, C) или* специальный функций *(например, при назначении подачи, частоты вращения и др.).*

G-адреса. Их используют, например, для программирования типа перемещения (с линейной или круговой интерполяцией. и др.). Слова с G-адресами относятся к числу инструкций, которые называют подготовительными функциями. Последние разбиты на группы, причем функции из разных групп взаимно независимы. С другой стороны, G-функции одной и той же группы взаимно модальны, т. е. действуют до отмены или замены G-функцией из той же группы. В кадре может быть представлена только одна G-функция из своей группы.

Адреса X, Y, Z, C и др. Эти адреса используют для обозначения координатных осей, вдоль которых осуществляются перемещения.

Пример: N G60 X10 Y10 B135.

Здесь X, Y – координатные оси подачи; B1 – ось вспомогательных перемещений.

Специальные функции. Примерами адресов специальных функций могут послужить: F (подача), S (частота вращения шпинделя), M (вспомогательная функция, связанная, например, с управлением электроавтоматикой), T (выбор инструмента). В примере показан кадр, в котором присутствуют позиционная информация и специальные функции G01, X40, Y50, F250, S500, T05, M03. Здесь задано перемещение X40, Y55 (траекторная информация), а также специальные функции: подачи F250, частоты вращения шпинделя по часовой стрелке S500 и функции инструмента T05б, обеспечивающей его доступность в инструментальном магазине.

Номера кадров. Именем кадра служит его номер. Имя состоит из адреса N и собственно номера (например, N10). Нумерация облегчает чтение программы. Принято нумеровать кадры последовательно, по возрастающей степени, с приращением 10 (например, N10 N20 N30 и т.д.). При этом возникает возможность включать дополнительные кадры при редактировании программы. При ветвлениях и переходах программы номера кадров служат метками. Они используются также в циклах и подпрограммах.

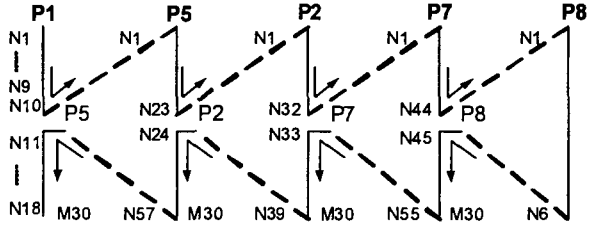
Комментарии. Комментарии необходимы для пояснений и документирования. Хорошо комментированная программа служит прообразом для других программистов при любых изменениях программы. Однако каждый символ комментария увеличивает длину файла управляющей программы на один байт. Комментарии указывают в скобках или предваряют кавычками. Комментарии в скобках игнорируются системой ЧПУ, а предваряемые кавычками визуализируются на экране монитора.

Работа управляющей программы. При отсутствии инструкций, управляющих потоком кадров, последние отрабатываются последовательно один за другим. Эта последовательность может быть нарушена инструкциями пропуска кадров, вызова подпрограмм, перехода к другим кадрам.

Если кадры программы помечены соответствующим образом (/), то система управления проигнорирует их, если активен сигнал Skip.

Подпрограммы. Если какая-то часть технологического процесса повторяется, ее целесообразно оформить в виде подпрограммы, которая вызывается по мере надобности. Существуют два способа вызова подпрограммы: с адресом P или без него. Синтаксис вызова подпрограммы с адресом P выглядит так: P<имя_подпрогр>DIN, где DIN означает, что все кадры подпрограммы написаны в коде DIN66025 (ISO6983), т. е. в коде ISO-7bit.

Все перемещения, заданные в том же кадре, будут выполнены до вызова подпрограммы. Последняя может иметь свои подпрограммы путем вложения (рис. 145).



P1 - главная программа
P5, P2, P7, P8 - подпрограммы

Рис. 145. Вложение подпрограмм

Подпрограммы могут быть также вызваны под G- и M-адресами (об этом далее). Вызов подпрограммы без использования адреса P осуществляется указанием ее имени. Кроме того, для вызова подпрограмм зарезервированы 16 G-функций.

Как правило, основная программа, кадры подпрограммы и циклы исполняются в том порядке, в каком они запрограммированы. Но он может быть нарушен переходами, условными и безусловными. Инструкции перехода зависят от конкретной системы ЧПУ и выходят за рамки стандарта DIN 66025 (ISO 6983).

5.3.2. Координатные оси и координатные системы

Физические и логические оси. Приводы станка относятся к приводам подачи и главного движения. Приводы подачи определяют положение в рабочем пространстве станка. Различают физические и логические координатные оси. Физические оси называют также системными. Они группируются по каналам ЧПУ, причем в рамках канала координатные оси находятся в единообразном технологическом отношении друг к другу. Таким образом, группы осей могут работать (выполнять технологические операции) независимо и параллельно. Физические оси, не привязанные к каналу, называют асинхронными или вспомогательными. Вспомогательные оси служат, к примеру, для организации перемещений в механизмах смены инструмента.

Отдельные оси внутри группы канала ЧПУ называют логическими. Они объединены интерполяционными алгоритмами, и в этой связи их также называют синхронными. Логические оси канала имеют индексы. Связывание физических и логических осей осуществляют при помощи так называемых «машинных параметров» станка.

Координатная система. Используют правоориентированную координатную систему, в которой предусмотрены линейные перемещения вдоль осей координат X, Y и Z. Каждая из этих осей может быть связана с круговыми вращениями поворотных осей A, B и C (рис. 146).

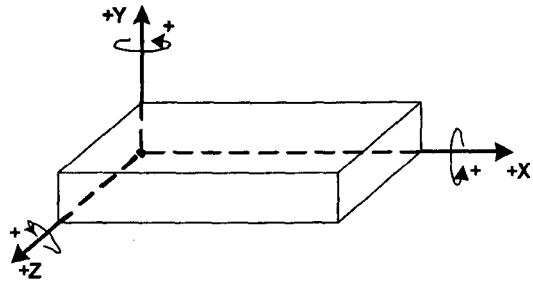


Рис. 146. Система координат и рабочее пространство станка

Если станок имеет единственный шпиндель, то ось Z параллельна оси шпинделя. В противном случае она перпендикулярна плоскости зажима детали. Положительные направления осей соответствуют относительному движению инструмента и заготовки. Ось X расположена в горизонтальной плоскости соответственно плоскости зажима заготовки. Аналогично располагается ось Y. Оси X, Y и Z являются главными. Кроме того, возможны параллельные управляемые оси, которым придают адреса U, V, W. Поворотные движения, привязанные к базовым координатам, имеют адреса A, B и C (рис. 147).

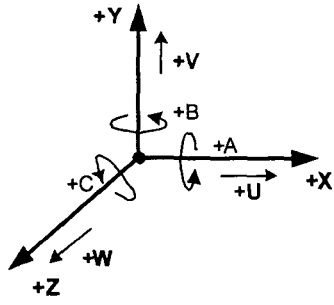


Рис. 147. Поворотные (A, B, C) и параллельные (U, V, W) оси станка

Положительное направление поворотных осей соответствует движению против часовой стрелки, если смотреть со стороны положительного направления оси.

Если существуют дополнительные параллельные координатные системы, то они имеют адреса P, Q и R.

Координатные системы. Для того чтобы исполнять управляющую программу без всяких изменений по отношению к чертежу, приходится определять несколько координатных систем. Переход от одной координатной системы к другой называется координатным.

Осевая координатная система ACS (Axes Coordinate System). Совокупность осей любого канала образует осевую координатную систему ACS. Заданное движение вдоль осевой координатной системы воспроизводится путем движения привода одной физической оси.

Машинная координатная система MCS (Machine Coordinate System). Осевая координатная система зависит от типа и кинематики технологической машины, а потому имеет небольшое значение при спецификации движений, связанных с обработкой деталей. По этой причине используют координатную систему MCS, привязанную к каналу. Как правило, эта система – декартова, а потому не зависит от кинематики технологической машины.

У каждого канала может быть своя машинная координатная система. Нулевую точку M системы называют машинной и обозначают ⊕.Отношение между осями машинной и осевой координатных систем называется осевой (или «обратной») трансформацией. На рис. 148 представлены примеры подобных отношений.

Относительную осевую точку R обозначают ⊕. Она необходима для установления связи между нулем машинной координатной системы и точкой автоматического выхода в нуль следящих приводов подачи, если позиционные датчики следящих приводов работают в относительной системе измерения. В этом случае приводы должны быть выведены в относительную точку при включении и выключении питания на станке. Такой необходимости нет, если приводы подачи располагают абсолютной измерительной системой.

Координатная система детали WCS (Workpiece Coordinate System). Ее назначают свободно в зоне машинной координатной системы. Нулевую точку координатной системы детали обозначают W и символом ⊕. Возможно определить несколько аддитивно связанных между собой координатных систем деталей.

Координатная система управляющей программы PCS (Program Coordinate System). Координатной системой управляющей программы PCS (рис. 149) называют такую координатную систему детали WCS, индекс нулевой точки которой имеет максимальное значение: W_i, где i = max. Нулевую точку этой системы обозначают P и символом ⊕. Все запрограммированные координаты управляющей программы соотносятся с нулевой точкой P. Координатную систему PCS можно свободно назначать и поворачивать в зоне системы WCS.

Координатная система инструмента TCS (Tool Coordinate System). Эта система определяет положение и ориентацию инструмента в машинной координатной системе. Нулевую точку системы обозначают T. Размеры инструмента (для трехкоординатного станка) задают по отношению к фиксированной точке, определяющей зажим инструмента. В разных случаях, показанных на рис. 150, точка T может совпадать с точками N или E.

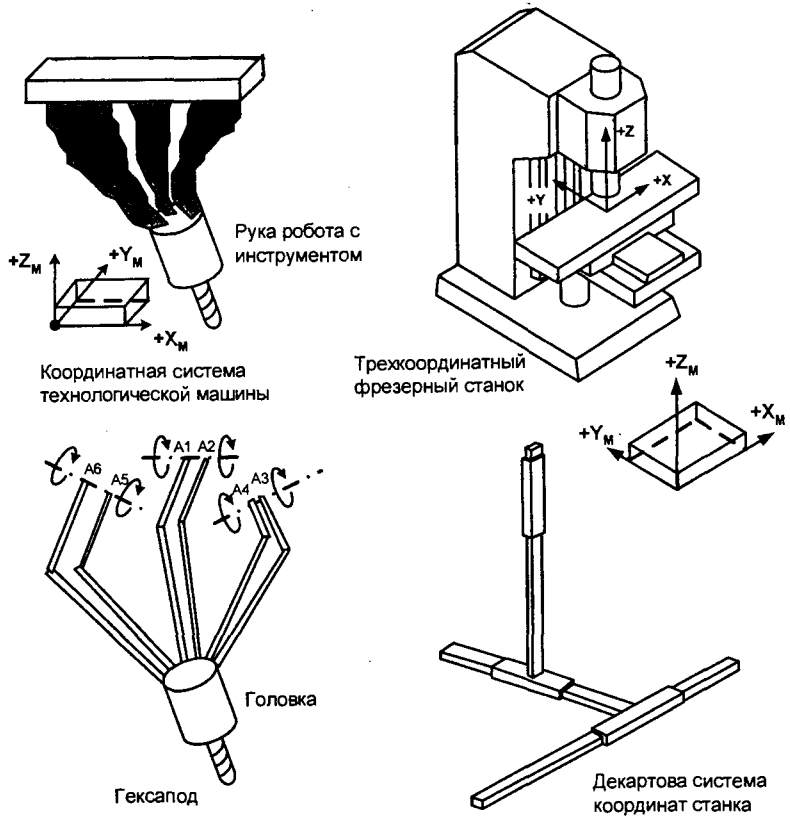


Рис. 148. Координатная система технологической машины и координатные системы осей

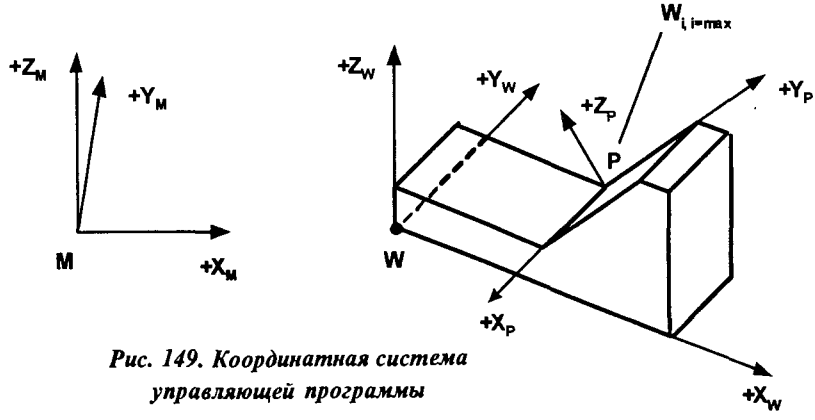


Рис. 149. Координатная система управляющей программы

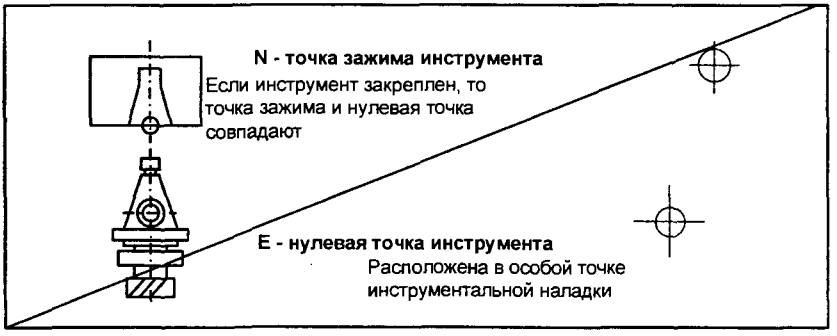


Рис. 150. Координатная система инструмента

Трансформация координат: машинных, детали и управляющей программы. Абсолютные значения координат обычно определены в системе MCS по отношению к нулевой точке M. Из практических соображений все размеры и перемещения, указанные в управляющей программе, заданы по отношению к нулевой точке P или W. При этом управляющие программы развязаны с машинными координатами. Благодаря программным смещениям, можно выполнять управляющую программу в любой зоне машинной системы координат без изменения размеров, указанных в программе. Если программные смещения отсутствуют, то все координаты управляющей программы интерпретируются как машинные.

Для программного смещения нуля координатной системы детали предусмотрены следующие инструкции:

- G53, G54,..., G59. Смещение нуля ZS (Zero Shift).
- G153, G154,..., G159. Первое аддитивное смещение нуля ZS.
- G253, G254,..., G259. Второе аддитивное смещение нуля ZS.
- G160, G260, G360, G167. Смещение нуля по внешней команде.

Положение детали может быть скорректировано путем смещения нуля ее координатной системы в плоскостях X/Y, X/Z, Y/Z и поворота в плоскости X/Y с помощью следующих инструкций:

- G138, G139. Коррекция (компенсация) положения детали.

Для коррекции положения детали путем смещения нуля ее координатной системы и поворотов в плоскостях X/Y, X/Z, Y/Z используют инструкции:

- G353, G354, G359. Наклон плоскости.
- G453, G454, G459. Первый аддитивный наклон.
- G553, G554, G559. Второй аддитивный наклон.

Как уже отмечалось, последней в серии координатных систем детали будет координатная система управляющей программы. При смещении ее нуля по отношению к координатной системе детали используют инструкции:

• G169, G168. Смещение нуля координатной системы управляющей программы.

• G269, G268. Аддитивное смещение нуля.

Применение отдельных инструкций показано на рис. 151.

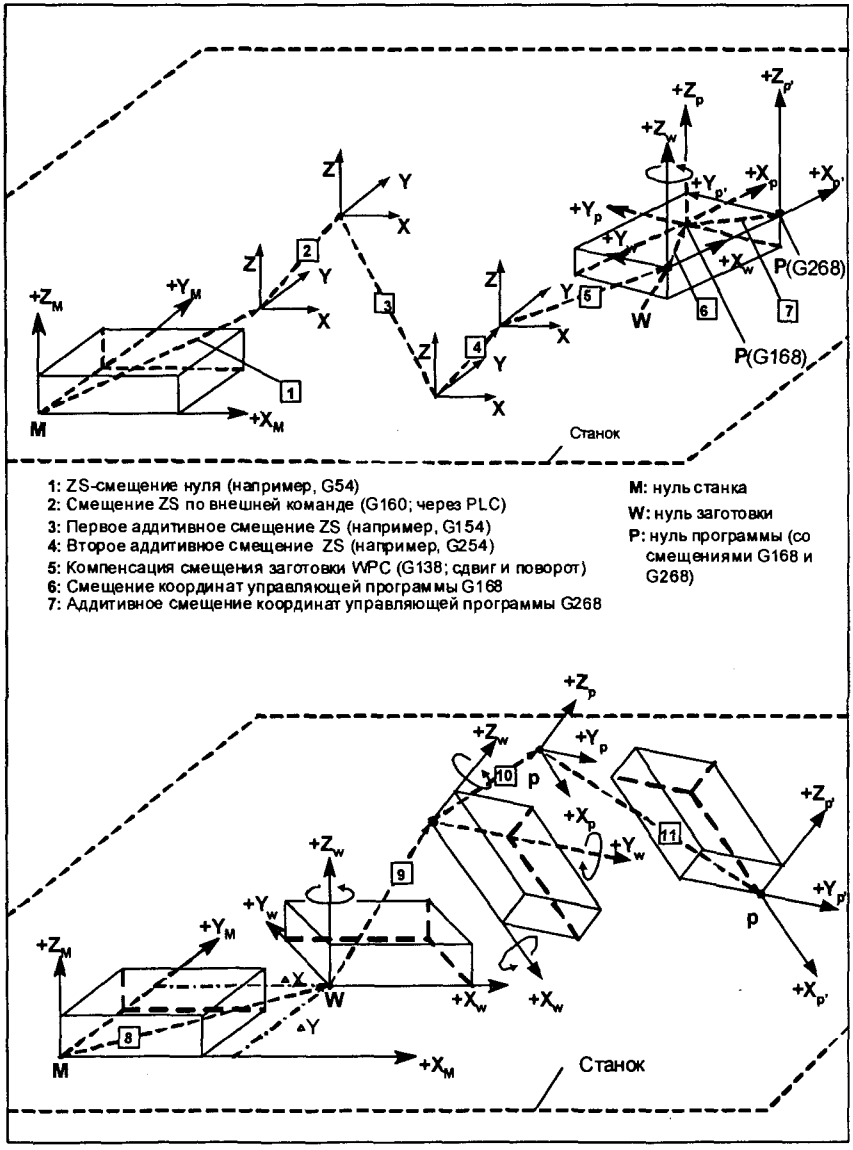


Рис. 151. Трансформация систем координат

Активизация смещений. Активизация смещений зависит от тех или иных G-функций; она осуществляется при помощи таблиц смещения нуля, а также первого и второго аддитивных смещений нуля ZS. Таблицы смещения нуля используют для хранения смещений между нулевыми точками M и P (или W).

Если соответствующее значение смещения активизировано, то это значение автоматически добавляется системой ЧПУ к каждому абсолютному значению координаты в управляющей программе. Таблицы смещения нуля представлены в файловой системе системы ЧПУ в форме ASCII файлов. Функция G22 активизирует эти таблицы в каждом канале.

Работа всех остальных G-функций рассмотрена в разделе программирования G-функций. Смещение нуля по внешней команде инициируется программируемым контроллером.

Процедура определения и сохранения смещений продемонстрирована на рис. 152. Сохранение осуществляется путем записи смещений в таблицу.

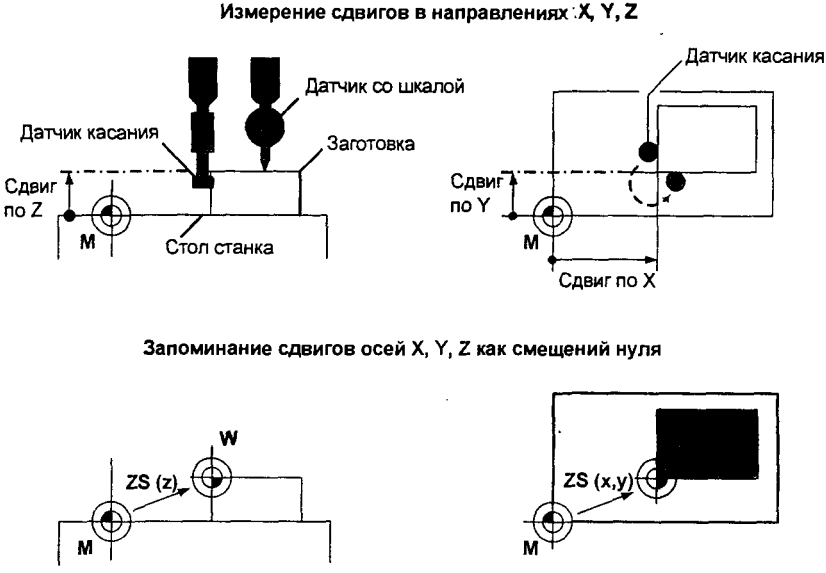


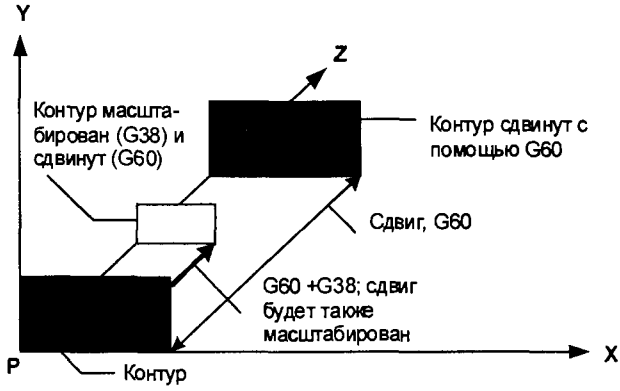
Рис. 152. Процедура определение и сохранение смещений

Функции манипулирования запрограммированным контуром. Возможны следующие функции манипулирования контуром:

- смещение (G60 – программирование смещения);
- зеркальное отображение, масштабирование, поворот вокруг оси, параллельной координатной оси (функции G37, G38).

Функции проиллюстрированы на рис. 153.

Операции с контуром, масштабирование и сдвиг: G60 + G38



Операции с контуром, масштабирование и сдвиг: сдвиг нуля + G38

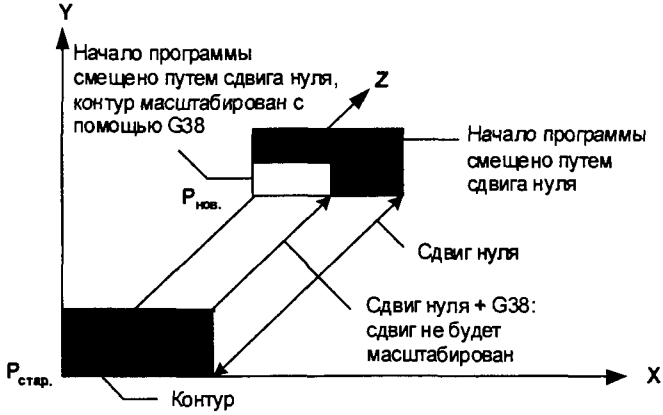


Рис. 153. Операции с контуром

Функции компенсации инструмента. Функцию инструмента обозначают адресом T и некоторым номером (например, слово T9 представляет собой инструмент номер 9). Инструментальный комплект состоит из инструмента и инструментальной державки.

В процессе обработки режущая кромка инструмента должна точно следовать вдоль запрограммированной траектории. В силу различия используемых инструментов их размеры должны быть учтены и введены в систему управления перед началом воспроизведения программы. Только в этом случае траектория может быть рассчитана безотносительно к параметрам

используемых инструментов. После того как инструмент установлен в шпиндель и активизирована соответствующая коррекция (компенсация его размеров), система ЧПУ автоматически принимает в расчет эту коррекцию.

Функции D и H компенсации инструмента. Функция H осуществляет компенсацию длины, а функция D – компенсацию радиуса (рис. 154).

Компенсация длины возможна двумя способами: по отношению к передней плоскости шпинделя (рис. 155) и к «нулевому» инструменту (рис. 156). В обоих случаях величины компенсации сохраняются в соответствующей таблице.

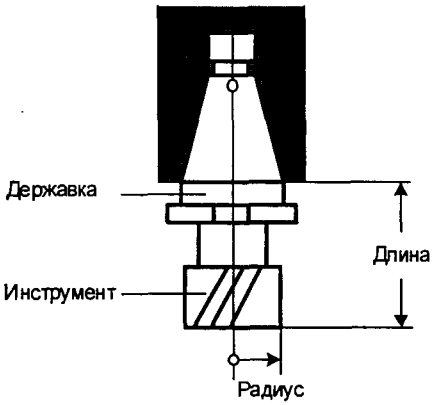


Рис. 154. Компенсация длины и радиуса инструмента

На рис. 155 для инструмента T01 H_1 =70,832 мм, для T02 H_2 = = 81,712 мм, для T03 H_3 = = 100,003 мм. Как видим, знак компенсации здесь может быть только положительным. Во втором случае выбирают «нулевой» инструмент, торцевая плоскость которого WSN (Workplane for Setting Null) служит для настройки и определения компенсации для всех остальных инструментов. «Нулевой» инструмент (T02 на рис. 156) имеет нулевое значение компенсации. Знак компенсации может быть положи-

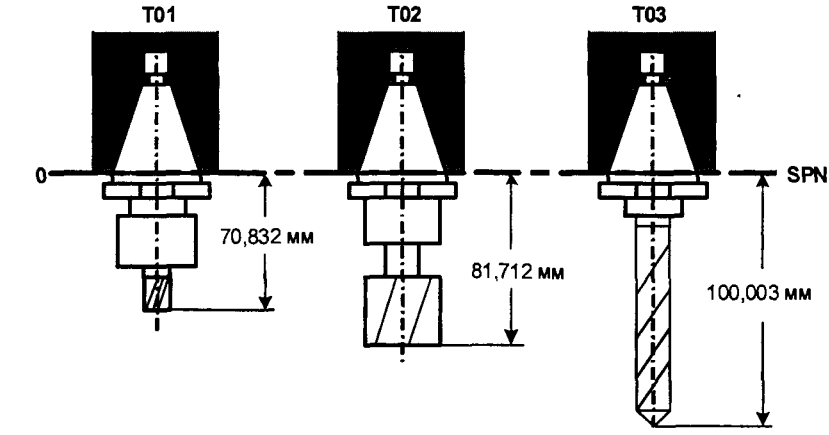


Рис. 155. Компенсация длины по отношению к передней плоскости шпинделя

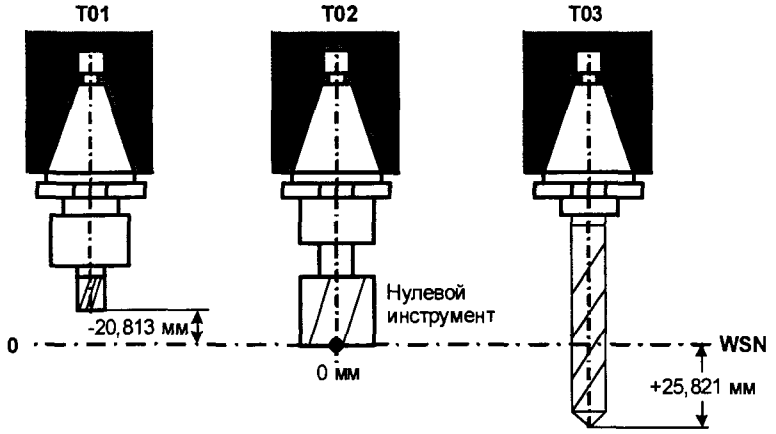


Рис. 156. Компенсация длины по отношению к «нулевому инструменту»

тельным или отрицательным. Например, для инструмента T01 H1=– 20, 813 мм, для T02 H2 = 0, для T03 H3 = 25,821 мм.

Центр фрезы движется по эквидистантной траектории, параллельной контуру детали, отстоящей от нее на величину, равную радиусу фрезы. Эквидистантную траекторию называют также траекторией центра фрезы. Значения компенсации для различных инструментов вносят в таблицу. Например, для T01 D1 = 14 мм (при диаметре фрезы 28 мм); для T02 D2 = 22 мм (при диаметре фрезы 44 мм). Детали эквидистантной коррекции (компенсации) будут рассмотрены при анализе G-инструкций G40, G41 и G42.

Внешняя компенсация инициируется программируемым контроллером с помощью инструкций G145 и G845.

Так называемая комплексная компенсация представляет собой набор компенсационных данных для 3D-коррекции инструмента или, например, для компенсации на длину инструмента в операциях с несколькими сверлами. Этот вид компенсации активизируется инструкциями G147 и G847. Комплексная компенсация может включать коррекцию на расположение режущей кромки.

5.3.3. Траектории движения (типы интерполяции)

Линейная интерполяция предполагает движение по прямой линии в трех-координатном пространстве. Перед началом интерполяционных расчетов система ЧПУ определяет длину пути на основе запрограммированных координат. В процессе движения производится контроль контурной подачи: ее величина не должна превышать допустимых значений. Движение по всем координатам должно завершиться одновременно.

При круговой интерполяции движение осуществляется по окружности в заданной рабочей плоскости. Параметры окружности (например, коор-

динаты конечной точки и ее центра) определяются до начала движения на основе запрограммированных координат. В процессе движения выполняется контроль контурной подачи – ее величина не должна превышать допустимых значений. Движение по всем координатам должно завершиться одновременно.

Винтовая интерполяция представляет собой комбинацию круговой и линейной инерполяций.

В процесс интерполяции вовлекаются синхронные координатные оси, например X, Y и Z. Вспомогательные (асинхронные) координатные оси в процесс интерполяции не вовлекаются. Примером движения вдоль асинхронной оси может служить позиционирование инструментального магазина. При задании скорости подачи асинхронной оси используют адрес FA.

Линейная интерполяция при ускоренном перемещении (G00). Эффект состоит в том, что запрограммированное перемещение интерполируется, а движение к конечной точке осуществляется по прямой линии с максимальной подачей. Скорость и ускорение подачи, по крайней мере одной оси, максимальны. Скорость подачи других осей контролируется таким образом, чтобы их движение завершилось в конечной точке одновременно. При активной инструкции G00 движение замедляется до нуля в каждом кадре. При этом выполнение «точного позиционирования» зависит от инструкций G161, G162. Если же необходимости в замедлении скорости подачи до нуля в конце каждого кадра нет, то вместо G00 используют G200. Значение максимальной скорости подачи не программируется, но задается так называемыми машинными параметрами в памяти системы ЧПУ. Инструкция G00 является модальной и деактивирует инструкции той же группы: G01 – G03, G05, G10-G13, G73, G200.

Линейная интерполяция на ускоренном перемещении без замедления до V = 0 (G200). Эффект состоит в том, что отсутствует замедление скорости подачи до нуля в конце каждого кадра, т. е. нет торможения на стыке соседних кадров, и процесс интерполяции продолжается. При этом должны соблюдаться предусловия: инструкции G61 и G163 пассивны. Если, тем не менее, инструкция G61 активна, то несмотря на инструкцию G200 торможение до нуля будет осуществляться в каждом кадре. Если же активна инструкция G163, то характер движения будет определяться функциями точного позиционирования (см. инструкции G164 – G166).

Значение максимальной скорости подачи не программируют, но задают машинными параметрами в памяти системы ЧПУ. Инструкция G200 является модальной и деактивирует инструкции той же группы: G00, G01, G02 – G05, G10 – G13, G73.

Линейная интерполяция с предусмотренной скоростью подачи (G01). Перемещение с заданной скоростью подачи (в F-слове) к конечной

точке кадра осуществляется по прямой линии (рис. 157). Все координатные оси завершают движение одновременно. Скорость подачи в конце кадра снижается до нуля, но только если инструкция G08 пассивна. Запрограммированная скорость подачи является контурной, т. е. значения подачи для каждой отдельной координатной оси будут меньше. Скорость подачи обычно ограничивают настройкой машинных параметров. Вариант комбинации слов с инструкцией G01 в кадре: G01_X_Y_Z_F_. Использование инструкции G01 имеет следующие особенности:

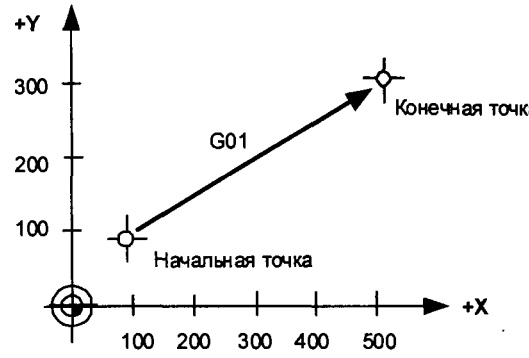


Рис. 157. Линейная интерполяция G01

- в любом кадре инструкция может быть представлена вместе с позиционными данными или без них;
- в любом кадре инструкция сопровождается F-словом, если до этого подача не была назначена;
- назначенная подача остается активной, пока ее значение не будет переопределено;
- инструкция является модальной и деактивирует инструкции той же группы: G00, G02, G03, G05, G10 – G13, G73, G200.

Фрагмент программы:

```
X100 Y100 /Начальное положение.
G01 X500 Y300 F100 /Движение к конечной точке.
```

Круговая интерполяция (G02, G03). Перемещение в кадре осуществляется по окружности с контурной скоростью, заданной в активном F-слове. Движение по всем координатным осям завершается в кадре одновременно также и в том случае, когда одна из осей не принадлежит плоскости круговой интерполяции. Вдоль этой оси движение будет линейно интерполируемым, а общая траектория станет винтовой линией. Инструкции G02 и G03 модальны и деактивируют другие G-инструкции той же группы. Приводы подачи задают перемещение по окружности с запрограммированной подачей в выбранной плоскости интерполяции, при этом G02 определяет движение по часовой стрелке, а G03 – против часовой стрелки. Выбор двух синхронных координатных осей осуществляется свободно путем выбора плоскости интерполяции.

При программировании окружность задают с помощью ее радиуса или координат ее центра. Дополнительная опция программирования окружности определяется инструкцией G05: круговая интерполяция с выходом на траекторию по касательной (см. далее).

Программирование окружности при помощи радиуса. Радиус всегда задают в относительных координатах, в отличие от конечной точки дуги, которая может быть задана как в относительных, так и в абсолютных координатах.

Используя положение начальной и конечной точек, а также значение радиуса, система ЧПУ прежде всего определяет координаты центра окружности. Результатом расчета могут стать координаты двух точек; ML

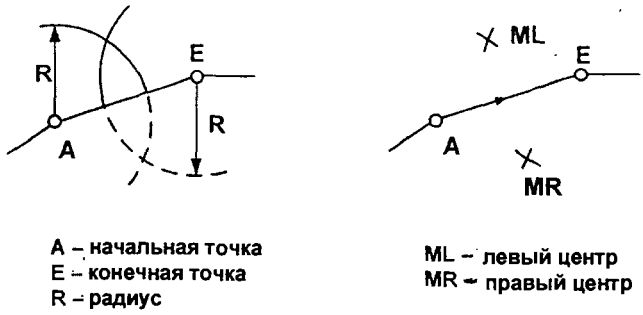


Рис. 158. Результаты расчета координат центров окружности

MR (рис. 158), расположенных соответственно слева и справа от прямой, соединяющей начальную и конечную точки.

Расположение центра окружности зависит от знака радиуса: при положительном радиусе центр будет находиться слева, а при отрицательном – справа. Расположение центра определяется также инструкцией G02 или G03 (рис. 159).

Как видно из рисунка, величина радиуса должна быть, по крайней мере, вдвое больше, чем длина отрезка, соединяющего начальную и конечную точки дуги окружности. Особым случаем является равенство отрезка удвоенному значению радиуса. Этот случай соответствует заданию полуокружности. Знак радиуса при этом значения не имеет.

Программирование полной окружности через задание радиуса недопустимо. Вариант комбинации слов с инструкцией G03 в кадре: N_G17_G03_X_Y_R±_F_S_M. Здесь инструкция G17 означает выбор круговой интерполяции в плоскости X/Y; инструкция G03 определяет круговую интерполяцию в направлении против часовой стрелки; X_Y_ пред-

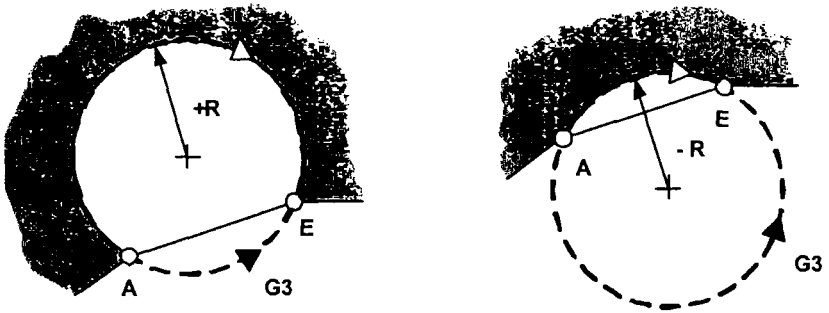


Рис. 159. Расположение центра окружности в зависимости от знака радиуса и инструкций G02 или G03

ставляют собой координаты конечной точки дуги окружности; R – радиус окружности.

Программирование окружности при помощи координат ее центра. Текущее положение используется в качестве начальной точки. Окружность, заданная координатами центра, проходит через начальную и конечную ее точки. Координатные оси, вовлеченные в процесс круговой интерполяции, имеют параметры I, J и K, приданные осям X, Y, Z соответственно. Параметры устанавливают расстояние между начальной точкой и центром M дуги окружности в направлении, параллельном осям. Знак определяется направлением вектора от A к M. Стандартное определение параметров указано на рис. 160.

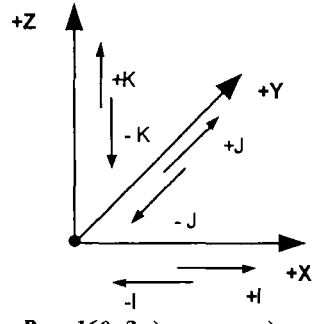


Рис. 160. Задание координат центра окружности

На рис. 160: I = M(X) – A(X); J = M(Y) – – A(Y); K = M(Z) – A(Z); I, J, K – параметры интерполяции; X, Y, Z – координатные оси, которым параметры I, J, K приданы соответственно; M – центр окружности, заданный относительно начальной точки дуги окружности.

На рис. 161–165 рассмотрены различные примеры программирования окружности.

***Пример 1*:**

N...G90 G17 G02 X350 Y25O I200 J-50 F...S...M...

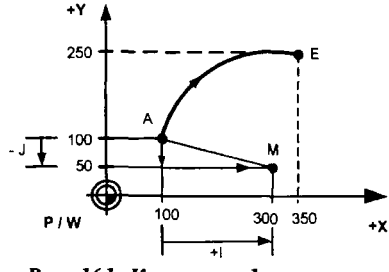


Рис. 161. К примеру 1 программирования окружности

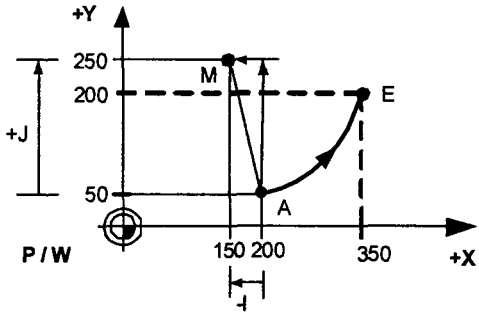


Рис. 162. К примеру 2 программирования окружности

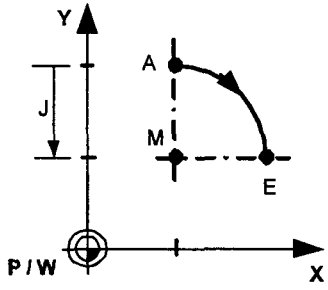


Рис. 163. К примеру 3 программирования окружности. Особенность: один из параметров интерполяции всегда равен нулю, и нет нужды упоминать его в программе. Здесь это параметр I

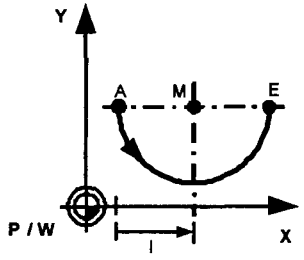


Рис. 164. К примеру 4 программирования окружности. Особенность: по оси Y координаты начальной и конечной точек совпадают. Перемещение по этой координате в кадре не указывают, как и параметр интерполяции J

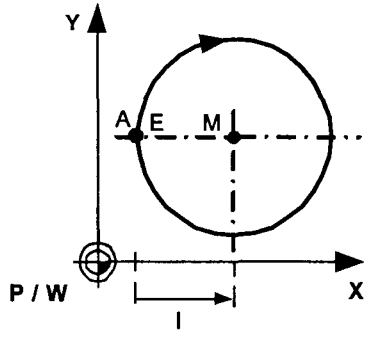


Рис. 165. К примеру 5 программирования окружности. Особенность: координаты начальной и конечной точек совпадают. Приращения по обеим координатам указывать в кадре не нужно. Если начальная и конечная точки лежат на границе квадрантов, то один из параметров интерполяции будет равен нулю, и его можно не указывать. Так, в приведенном примере могут быть опущены функции X, Y и J

Пример 2:
N...G90 G17 G03 X350 Y200 I-50 J200 F...S...M...
Пример 3 (программирование четверти окружности):
N...G17 G02 X...Y...J-...F...S...M...
Пример 4 (программирование полуокружности):
N...G17 G03 X...I...F...S...M...
Пример 5 (программирование полной окружности):
N...G17...G02 I...F...S...M...

Винтовая N-интерполяция (G202, G203). В процессе винтовой N-интерполяции осуществляется круговая интерполяция в выбранной плоскости и линейная интерполяция для остальных синхронных координатных осей общим числом до шести круговых или линейных осей. Это связано с тем, что общее число синхронных осей в одном канале не превышает восьми. Движение по всем координатам завершается одновременно.

Винтовая N-интерполяция является обобщением простой винтовой, при которой линейная интерполяция осуществляется только для одной оси, перпендикулярной выбранной плоскости круговой интерполяции.

Плоскость круговой интерполяции определяется инструкциями G17, G18, G19, G20. В одном кадре может быть запрограммирована только одна полная окружность. Скорость подачи является контурной, однако есть некоторые особенности для линейно интерполируемых осей, связанные с использованием инструкций G594 и G595.

Движение по окружности по часовой стрелке осуществляется соответственно инструкции G202, а против часовой стрелки – согласно инструкции G203. Программирование окружности возможно с использованием радиуса и координат центра окружности.

Инструкция винтовой интерполяции является модальной и принадлежит ко второй группе модальных G-инструкций. На рис. 166 приведен пример простой винтовой интерполяции:

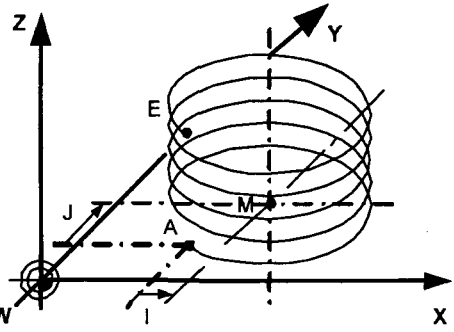


Рис. 166. Пример простой винтовой интерполяции. Особенность: координаты начальной и конечной точек совпадают в плоскости координат X и Y. Указывать параметр интерполяции K не следует

N...G91 G17 G03 X...Y...Z...I...J...F...S...M...

Круговая (винтовая) интерполяция с выходом на круговую траекторию по касательной (G05). Система ЧПУ использует инструкцию G05 для расчета такого кругового участка, выход на который из предыдущего кадра (с линейной или круговой интерполяцией) осуществляется по касательной. Параметры формируемой дуги определяются автоматически, т.е. программируется только ее конечная точка, а радиус не задается: G5 X... Y... Различные примеры программирования с инструкцией G05 показаны на рис. 167.

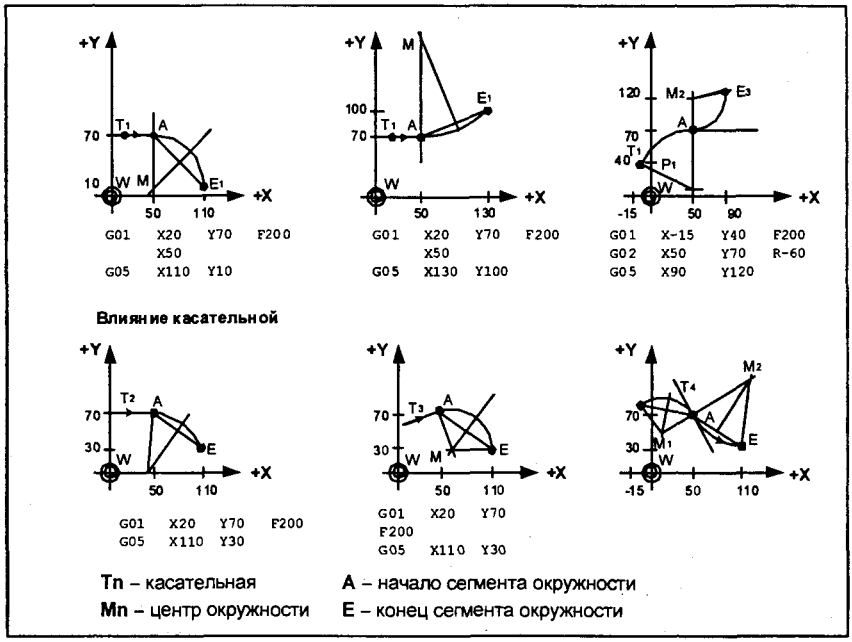


Рис. 167. Круговая (винтовая) интерполяция с выходом на круговую траекторию по касательной

5.3.4. Группирование координатных осей (G581, G580)

Группирование осей приводит к жесткому позиционному соотношению между ведущей и ведомыми осями. Каждая группа состоит из одной ведущей оси и до семи ведомых осей. Группа осей работает в одном и том же канале системы ЧПУ. За каждым каналом (в многоканальных системах ЧПУ) может быть закреплено несколько групп осей (рис. 168). Инструкция G581 служит для создания таких групп, а инструкция G580 – для их расформирования.

Существуют следующие варианты групп осей:

- параллельные (например, если несколько исполнительных органов перемещаются параллельно), электронные «гитары» (с осями, которые связаны определенным передаточным отношением);

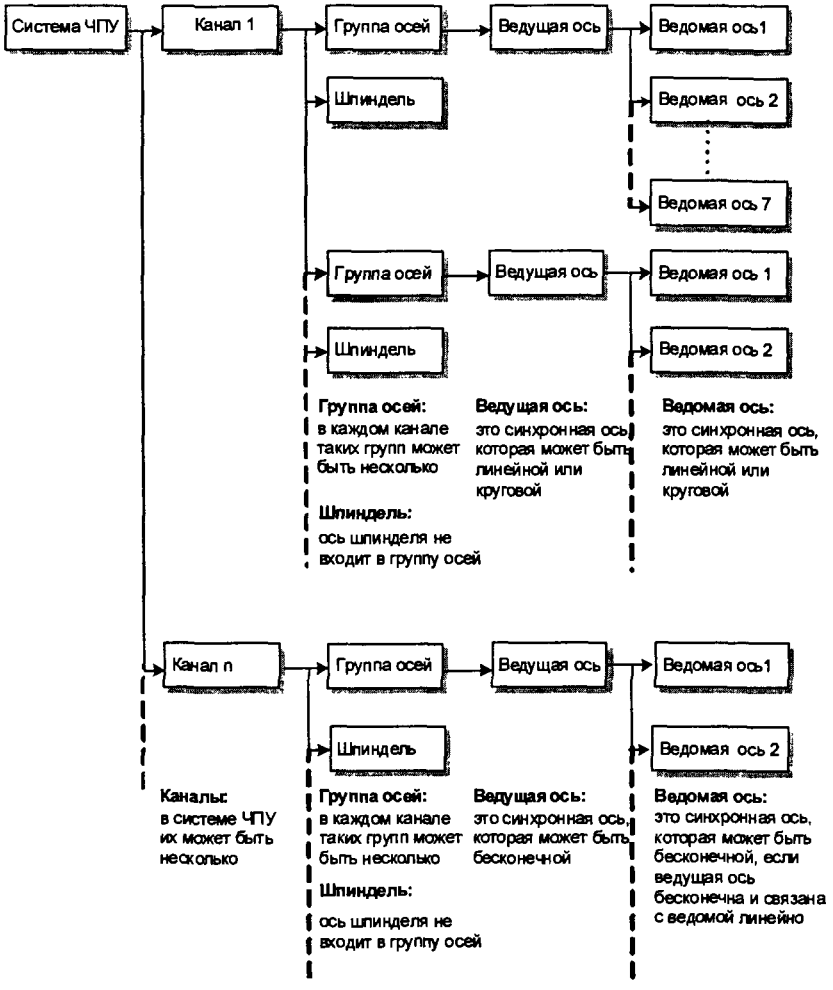


Рис. 168. Группирование координатных осей

- нелинейно-связанные.

Характеристикой группы служит отношение ведомых осей к ведущей. Линейное отношение связывает положение ведущей оси p_m с положением p_s ведомой оси:

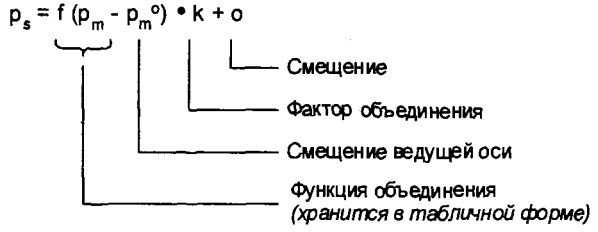
$$p_s = p_m \cdot k + o$$

Смещение

Фактор объединения

Здесь k = 1 для параллельных осей, для электронных «гитар» значение k определяется настройкой электронной «гитары».

Нелинейно-связанные оси представлены функцией $f(p_m)$ (рис. 169), которая хранится в табличной форме в файловой системе системы ЧПУ:



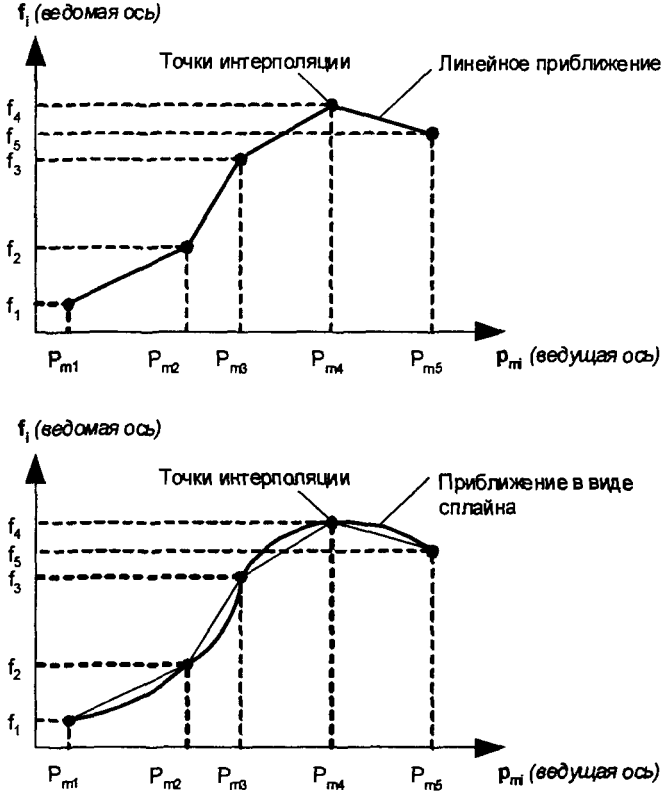


Рис. 169. Нелинейно связанные оси: линейное приближение и приближение в виде сплайна

5.3.5. Управление шпинделем

Функции шпинделя. Функции могут относиться к отдельным шпинделям или шпиндельным группам. Максимальное число шпинделей равно восьми, и каждый из них может быть придан любой из четырех предусмотренных шпиндельных групп с помощью машинных параметров. Примеры отношений вспомогательных М-функций и шпинделей: М03 относится к первой шпиндельной группе, М103 – к первому шпинделю, М203 – ко второму шпинделю. Все эти вспомогательные функции включают вращение шпинделя (или шпиндельной группы) по часовой стрелке.

Аналогичным образом, вспомогательные функции М13, М113, М213 включают вращение шпинделя (или шпиндельной группы) по часовой стрелке с одновременной активизацией функции охлаждения. Вспомогательные функции М04, М104, М204 включают вращение шпинделя (или шпиндельной группы) против часовой стрелки. Аналогичным образом вспомогательные функции М14, М114, М214 включают вращение шпинделя (или шпиндельной группы) против часовой стрелки с одновременной активизацией функции охлаждения. Вспомогательные функции М05, М105, М205 останавливают вращение шпинделя.

Ориентированная остановка шпинделя (шпиндельной группы). Вспомогательные функции М19 (для первой шпиндельной группы), М119 (для первого шпинделя), М219 (для второго шпинделя) служат для программирования ориентированной остановки вращения шпинделя (или шпиндельной группы). При этом может быть использовано или не использовано S-слово. Если S-слово не используется, то шпиндель останавливается по углу в своей относительной точке. При использовании S-слова указывают угол позиционирования в градусах по отношению к относительной точке шпинделя.

Пример:

```
N...M19 /Шпиндели первой группы устанавливаются соответственно /
в свои относительные точки.
N...M119 /Первый шпиндель устанавливается в свою относительную /
точку.
N...M219 /Второй шпиндель устанавливается в свою относительную /
точку.
N...M19 S180 /Шпиндели первой группы устанавливаются под углом 180 /
градусов к относительной точке.
N...M119 S1=180 /Первый шпиндель устанавливается под углом 180 /
градусов к относительной точке.
```

Использование шпиндельной бабки с зубчатыми передачами. Общий диапазон регулирования частоты вращения шпинделя разбивается с помощью шпиндельной бабки на несколько поддиапазонов (не более че-

тырех). Все характеристики этих поддиапазонов отражаются в машинных параметрах. Для активизации автоматического переключения в шпиндельной бабке используются вспомогательные функции M40, M140 и M240, различие между которыми такое же, как и в рассмотренных выше шпиндельных функциях.

Пример:

```
N...M40 /Включение автоматического выбора поддиапазона для первой /
шпиндельной группы.
N...M140 /Включение автоматического выбора поддиапазона /
для первого шпинделя.
N...M240 /Включение автоматического выбора поддиапазона /
для второго шпинделя.
```

Программирование частоты вращения. Частота вращения программируется для отдельного шпинделя или всех шпинделей группы с помощью S-слова. Варианты использования слова таковы:

«Si=» означает, что программируется частота вращения для шпинделя под номером «i»; «SSPGj = » означает, что программируется частота вращения для шпиндельной группы под номером «j»; использование только лишь адреса S означает, что программируется частота вращения шпинделя той группы, которой по умолчанию принадлежит первый шпиндель.

Пример:

```
N...G97 /Активизация программирования частоты вращения шпинделя.
N...G...X...Y...Z...F...SSPG1=1000 /Частота вращения шпинделей/
первой группы равна1000 об/мин.
N...G...X...Y...Z...F...S1=2000 /Частота вращения первого шпинделя /
равна 2000 об/мин.
N...G...X...Y...Z...F...S3=2000 /Частота вращения третьего /
шпинделя равна 2000 об/мин.
N...G...X...Y...Z...F...S1500 /Первый шпиндель из той группы, которой
/он принадлежит по умолчанию, имеет частоту вращения 1500 об/мин.
```

5.4. Методика разработки управляющей программы ЧПУ соответственно стандарту ISO 14649 STEP-NC

Построение управляющей программы ЧПУ соответственно стандарту ISO 14649 STEP-NC требует использования модели, особенности которой состоят в однозначном соответствии EXPRESS-модели и предполагает удобное древовидное представление экземпляра программы на экране системы ЧПУ. В основе такой модели лежит язык XML, а ее разработка поддержана ин-

струментальными средствами XML Spy фирмы Altova. Построенная модель проиллюстрирована описаниями EXPRESS-сущностей и соответствующими XML-схемами. Представлены фрагмент документа модели и текстовый фрагмент управляющей программы на языке XML.

Стандарт ISO 14649 STEP-NC предлагает модель того, ***что*** нужно сделать для представления информации на уровне системы ЧПУ, но не подробности того, ***как*** осуществлять траекторные перемещения и выполнять команды управления электроавтоматикой [86, 87]. Модель того, что нужно сделать, представлена в управляющей программе специального вида на языке EXPRESS, который неудобен для непосредственной трансляции в терминальном модуле системы ЧПУ. В этой связи наше внимание привлек язык XML (Extensible Markup Language), который обладает для целей ЧПУ двумя несомненными достоинствами. Во-первых, он позволяет построить модель, имеющую однозначное соответствие с EXPRESS-моделью, во-вторых, построенная модель имеет наглядное внешнее (для экрана системы ЧПУ) древовидное представление и удобную внутреннюю форму хранения данных в DOM-форме (Document Object Model).

Разработка внешнего представления составляет первую фазу жизненного цикла управляющей программы в стандарте ISO 14649, а создание внутреннего представления составляет вторую фазу жизненного цикла и является, по сути, разработкой входного модуля системы ЧПУ. Эта статья посвящена разработке методики для первой фазы жизненного цикла. При этом использована инструментальная система XML Spy фирмы Altova.

5.4.1. Инструментальная система XML Spy

XML-документ удобно проектировать в среде инструментальной системы XML Spy в графической форме на языке «схем» XSDL (XML Schema Definition Language) [88, 89]. Готовые схемы могут быть отредактированы, документированы и конвертированы в XSL (Extensible Stylesheet Language) или XML – формат. Окончательный XML-документ может быть представлен в текстовой форме (Word или HTML). Проектирование схем сопровождается назначением атрибутов и присвоением типов. Схема (schema) задает синтаксическую структуру документа с элементами семантики и включением данных, однако обладает несопоставимо (в сравнении с любыми другими синтаксическими определениями) большей наглядностью и информативностью.

Покажем элементы схем в упрощенной форме на примере обычной управляющей программы ЧПУ в стандарте ISO 6983 (рис. 170).

Из приведенного примера видно, что схемы представляют собой последовательно раскрываемую иерархию. Каждая схема состоит из одной

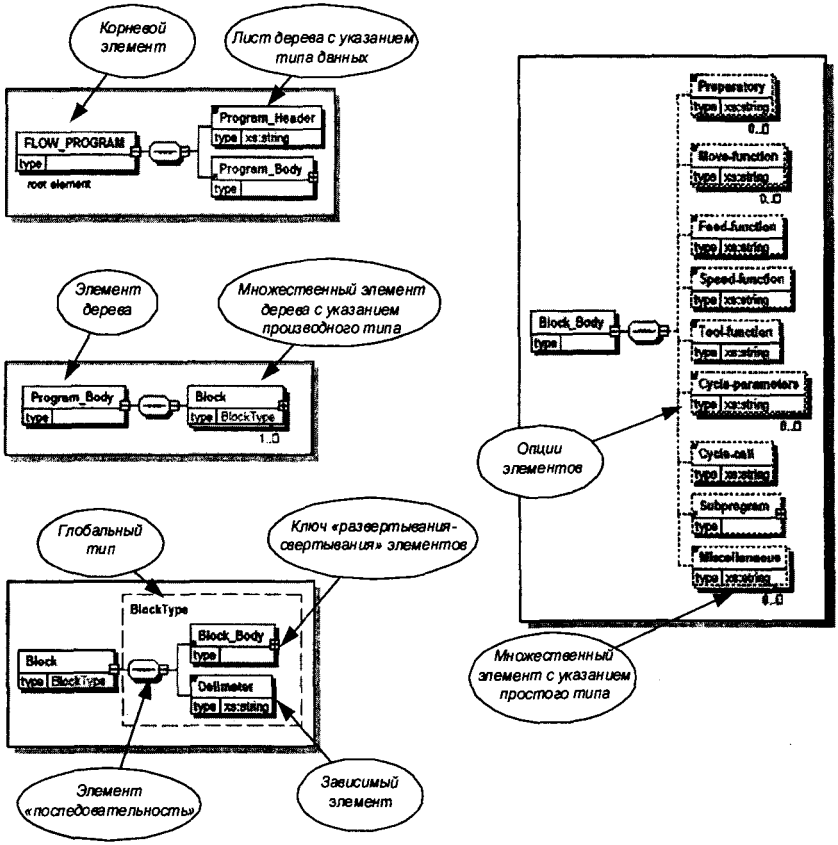


Рис. 170. Схемы традиционной управляющей программы ЧПУ

или нескольких ветвей с родительскими и дочерними элементами. Ветви завершаются листьями, в которых сосредоточены данные.

5.4.2. Схемы управляющей программы в стандарте STEP-NC

Все элементы управляющей программы описаны в стандарте ISO 14649 STEP-NC на языке EXPRESS. Эта информация является базовой при построении схем. При этом атрибуты EXPRESS-модели могут стать элементами схемы.

Исполняемый блок *executable* является базовым для всех других исполняемых объектов. Исполняемые блоки *executables*, выстроенные в определенном порядке, инициируют активность станка. Существуют три типа

исполняемых блоков: план операции *workplan*, функция ЧПУ nc_function, шаг операции *workingstep*.

```
ENTITY executable
ABSTRACT SUPERTYPE OF (ONEOF(workingstep, nc_function,
program_structure));
its_id: identifier;
END_ENTITY;
```

Поле объекта *executable*:

- its_id – идентификатор исполняемого блока. Он должен быть уникальным в пределах управляющей программы.

Структура управляющей программы *program structure* используется для построения логических блоков в рамках структурного программирования обработки и предопределяет ее выполнение. Шаг операции *workingstep* описывает технологические или вспомогательные процессы обработки, в которые вовлечены интерполируемые оси. ЧПУ-функции *nc_function* отображают процессы, единичные события и другие функции, не связанные с интерполяцией. Схема исполняемого блока *executable* приведена на рис. 171.

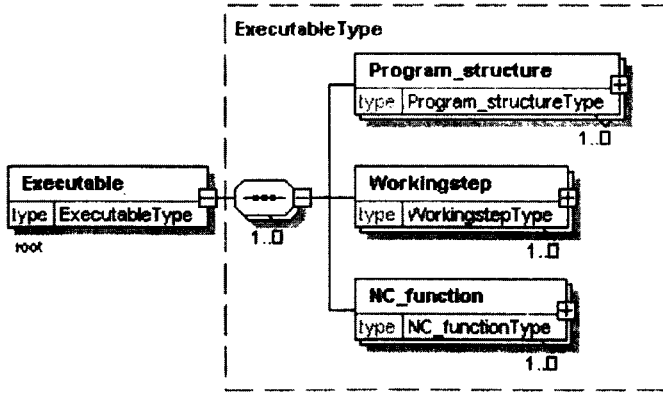


Рис. 171. Схема исполняемого блока executable

Структура управляющей программы program structure является исполняемым объектом, который включает другие исполняемые блоки executables. Включенные объекты executables могут выполняться в зависимости от условий или так, как это определено объектом program structure:

```
ENTITY program_structure
ABSTRACT SUPERTYPE OF (ONEOF(workplan, parallel, non_sequential,
selective, if_statement, while_statement, assignment))
SUBTYPE OF (executable);
END_ENTITY;
```

Схема объекта *program structure* представлена на рис. 172.

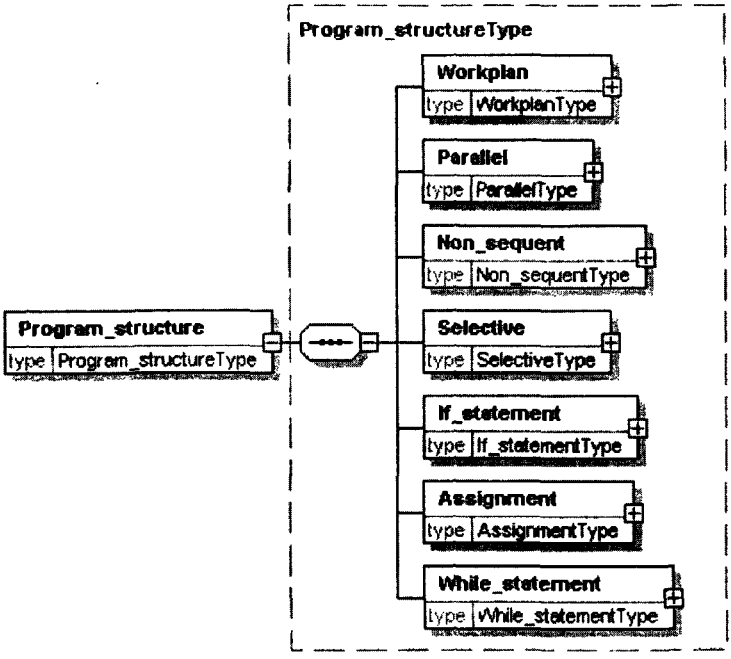


Рис. 172. Схема объекта program structure

План операции *workplan* объединяет несколько шагов операции *workingsteps* и NC-функции в линейном порядке:

```
ENTITY workplan
SUBTYPE OF (program_structure);
its_elements: LIST[1:?] OF executable;
its_channel: OPTIONAL channel;
its_setup: OPTIONAL setup;
its_effect: OPTIONAL in_process_geometry;
WHERE
WR1: SIZEOF(QUERY(it <* its_elements | it = SELF)) = 0;
END_ENTITY;
```

Описание полей плана операции *workplan* см. в разд. 2.6.2.

Схема плана операции *workplan* приведена на рис. 173.

Шаг операции *workingstep* является важнейшим строительным блоком управляющей программы ЧПУ в стандарте ISO 14649:

```
ENTITY workingstep
```

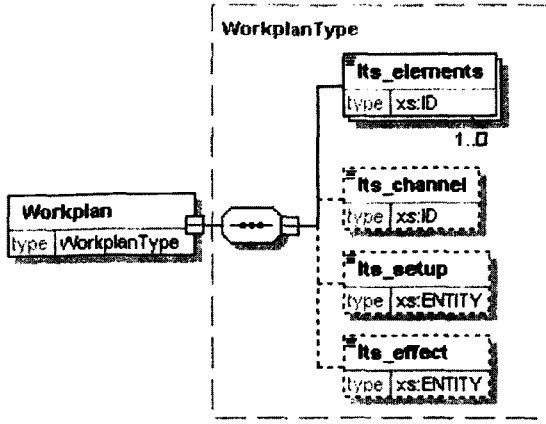


Рис. 173. Схема плана операции workplan

```
ABSTRACT SUPERTYPE OF (ONEOF (machining_workingstep,
rapid_movement,touch_probing))
SUBTYPE OF (executable);
its_secplane : elementary_surface;
END_ENTITY;
```

Поля шага операции *workingstep*:

- its_secplane: безопасная плоскость для шага операции *workingstep*. На этой плоскости или выше нее возможны безопасные, т. е. без коллизий, движения инструмента. Размеры заданы в координатной системе заготовки или, если шаг операции *workingstep* ассоциирован с типовой обрабатываемой формой manufacturing_feature, в локальной системе координат типовой формы *feature*.

Блоки могут быть как технологически нейтральными действиями (ускоренными перемещениями, измерительными циклами), так и рабочими, связанными с технологиями фрезерования, сверления, точения и др. Реальное содержание *workingstep* специфицировано в объекте «переход» *operation* и соответствующих подтипах. Существует возможность повторного использования информации *operation* (но не шагов операции) для нескольких типовых обрабатываемых форм *features* заготовки.

Переход *operation* может быть ассоциирован со многими формами *features* и использован в разных местах. С другой же стороны, шаг операции *workingstep* уникален. Дублирование егов пределах плана операции *workplan* в точности воспроизведет те же самые действия станка.

Шаги операции *workingsteps* описывают технологические или другие переходы, в которые обязательно вовлечены интерполируемые оси. Они описывают не последовательность событий во времени, а скорее условия

и состояния, при которых переход может быть выполнен. К примеру, шаг *workingstep* может вызывать определенный инструмент, который требуется для выполнения перехода, а также специфицировать использование охлаждения. Или иначе: может точно устанавливать, в какой момент должна быть произведена смена инструмента и что должно произойти раньше – смена инструмента или включение охлаждениея. Решение принимает программист или система ЧПУ в реальном времени. Например, если требуемый инструмент уже находится в шпинделе со времени предыдущего шага *workingstep*, система ЧПУ примет решение не менять инструмент.

Технологические шаги операции *machining workingsteps* представляют технологический процесс в определенной области заготовки. В отличие от других технологические шаги операции не существуют без типовых обрабатываемых форм *features*. По существу, они специфицируют отношение между определенной формой *feature* и переходом, который должен быть выполнен над формой *feature*. При этом устраняется двусмысленность отношения n : 1 между *features* и *operations*, а недвусмысленность спецификаций создает условия для работы станка.

Как и рассмотренный ниже переход *operation*, технологический шаг операции характеризуется единственным инструментом и набором технологических параметров, которые обычно остаются неизменными в рамках технологического шага операции. Его выполнение, если это необходимо, начинается со смены инструмента. В процессе реализации технологического шага *machining workingstep* смена инструмента невозможна. Станок должен достичь условий выполнения перехода its_operation (с атрибутами its_tool и its_technology), прежде чем состоится шаг операции *workingstep*. Если условия не выполнены в рамках предыдущего перехода, то перед очередным шагом *workingstep* произойдет остановка (например, после ускоренного перемещения).

Технологический шаг операции определен следующим образом:

```
ENTITY machining_workingstep
SUBTYPE OF (workingstep);
its_feature: manufacturing_feature;
its_operation: machining_operation;
its_effect: OPTIONAL in_process_geometry;
END_ENTITY;
```

Описание полей технологического шага операции см. в разд. 2.6.2.

Замечание. В результате интерпретации перехода *operation* технологический шаг операции имеет обычно четко обозначенные начальную и конечную точки. По этой причине два технологических шага операции не могут следовать один за другим в плане операции *workplan*. Разрыв между

конечной точкой предшественника и начальной точкой последователя вызовет остановку станка. Из этого правила существуют три исключения:

- конечная и начальная точки смежных технологических шагов операции совпадают;
- между смежными технологическими шагами операции необходима смена инструмента, причем система ЧПУ должна сама рассчитать траектории к позиции смены инструмента и от этой позиции;
- определение стартовой позиции для технологического шага операции оставлено за системой ЧПУ.

Схема технологического шага *machining workingstep* приведена на рис. 174.

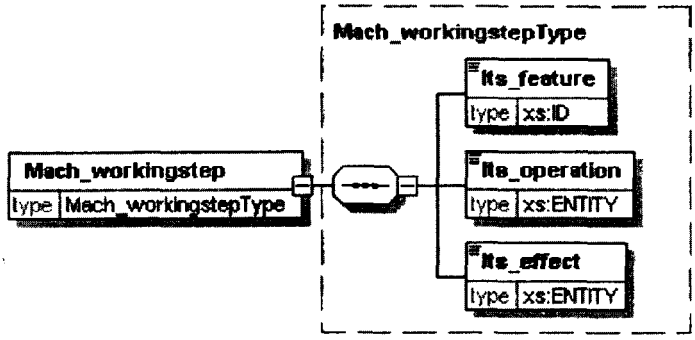


Рис. 174. Схема технологического шага machining workingstep

На самом нижнем информационном уровне, согласно стандарту ISO 14649, находятся переходы *operations*, в которых среди прочего описана траектория инструмента, если этого требует система CAM или ЧПУ.

Переходы *operation* специфицируют общие данные, которые им необходимы. Все переходы имеют опцию, специфицирующую точную траекторию инструмента. Для 2D обработки многие системы ЧПУ специфицируют циклы, в рамках которых генерируются собственные траектории инструмента. Менее продвинутые системы ЧПУ не в состоянии самостоятельно генерировать траектории. В тех случаях, когда они нуждаются в четко заданных траекториях, может быть использован атрибут «список траекторий» its_toolpath.

```
ENTITY operation
ABSTRACT SUPERTYPE OF (ONEOF (machining_operation,
rapid_movement,
touch_probing));
its_toolpath: OPTIONAL toolpath_list;
its_tool_direction: OPTIONAL tool_direction;
END_ENTITY;
```

- its_toolpath: список всех траекторий в данном переходе,
- its_tool_direction: обозначение ориентации инструмента. Если ориентация не указана, то используется по умолчанию та, которая специфична для текущей технологии.

Отметим, что переход *operation* геометрически связан с ассоциированной типовой обрабатываемой формой manufacturing_feature, если подобная ассоциация существует. Если форма *feature* имеет атрибут feature_placement, то соответствующие геометрические преобразования будут применены ко всем используемым размерным характеристикам: речь идет о списке траекторий its_toolpath и другой геометрической информации в подтипах перехода, такой как, например, направления и ориентации. Другими словами, геометрия и параметры технологического перехода machining_operation определены в локальной координатной системе типовой обрабатываемой формы *feature*.

Для типовых обрабатываемых форм *features* без атрибута feature_placement или для переходов *operations* без ассоциированной формы *feature* все данные интерпретируются в координатной системе заготовки, как это указано в атрибуте its_origin плана операции, использующего переход.

Схема перехода *operation* приведена на рис. 175.

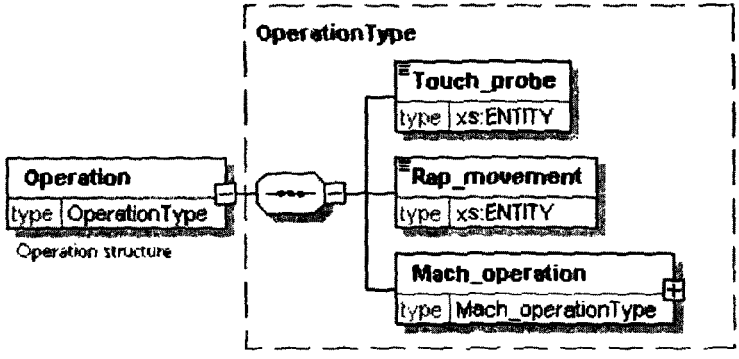


Рис. 175. Схема перехода operation

Технологический переход machining *operation* задает технологический процесс в ограниченной области заготовки и составляет содержимое технологического шага операции *workingstep*. По отношению к соответствующей типовой обрабатываемой форме *feature* переход специфицирует как минимум инструмент и набор технологических параметров. Эти данные составляют интегрированную часть перехода и, как правило, не могут быть изменены в области его определения. Однако для специальных переходов, когда определена точная траектория инструмента, можно использовать кон-

кретные параметры, изменяемые в пределах части траектории. В этом случае данные, представленные в переходе machining_operation, служат данными по умолчанию, если специфическая технологическая информация для траектории инструмента отсутствует.

Начальным движением инструмента в пределах machining_operation обычно является подвод к заданной стартовой точке. Она может быть явно определена на траектории инструмента или с помощью параметрически заданного пути, или оставлена на усмотрение системы ЧПУ. Последним движением инструмента является его отвод или подъем. Траектория в пределах перехода machining_operation, если не задана точно, будет определяться исходя из технологической стратегии и дополнительных параметров, заданных в подтипах перехода machining_operation.

Переход machining_operation является суперклассом всех технологий, включенных в стандарт ISO 14649. Для каждой технологии в специфичных технологически зависимых разделах определена стратегия обработки. Эта стратегия может служить исходной информацией для системы ЧПУ: как генерировать траектории, если они явно не заданы.

```
ENTITY machining_operation
ABSTRACT SUPERTYPE
SUBTYPE OF (operation);
its_id: identifier;
retract_plane: OPTIONAL length_measure;
start_point : OPTIONAL cartesian_point;
its_tool: machining_tool;
its_technology: technology;
its_machine_functions: machine_functions;
(*Informal proposition: If attribute SELF\operation.its_toolpath exists, then
attributes its_machining_strategy, retract_plane and start_point, if present,
are for information only.*)
END_ENTITY;
```

- its_id: уникальный идентификатор перехода;
- retract_plane: высота плоскости отвода, ассоциированной с переходом, задает начальную и конечную точки перехода по оси Z. Движения от этой плоскости к начальной точке резания и от последней точки к плоскости отвода осуществляются на рабочей подаче перехода. Ускоренные движения rapid_movement происходят только выше этой плоскости и только в новом переходе. Если плоскость и наследуемый атрибут its_toolpath не заданы, система ЧПУ сама определяет эту плоскость, которая в этом случае может быть плоскостью *security plane*;
- start_point: начальная точка процесса резания, которая определена как центр инструмента в локальной плоскости xy. plane;

- its_tool: инструмент, используемый в данном шаге операции;
- its_technology: технологические параметры перехода (частота вращения шпинделя, подача);
- its_machine_functions: указывает состояния технологических параметров, таких как охлаждение, удаление стружки и др.

Схема перехода *machining operation* представлена на рис. 176.

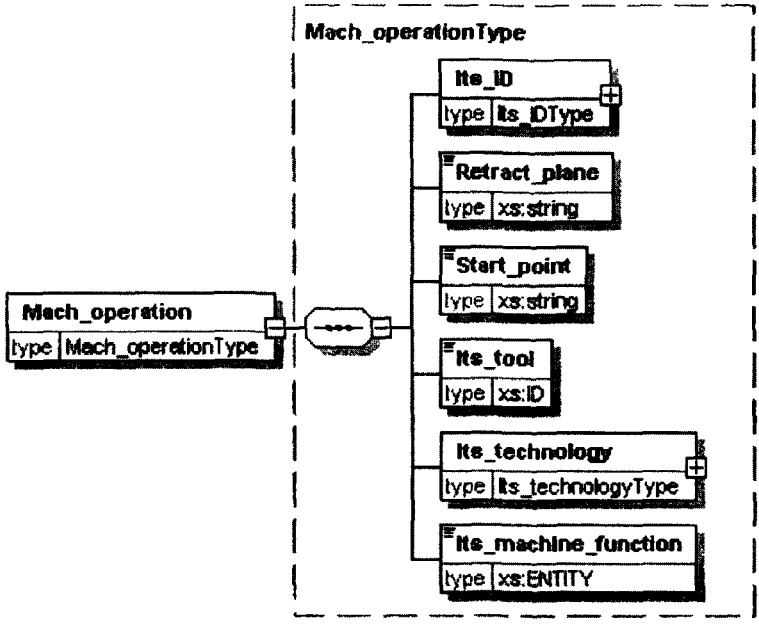


Рис. 176. Схема технологического перехода machining operation

Выше были приведены отдельные фрагменты схем управляющей программы в стандарте ISO 14649 STEP-NC. В заключение первой фазы жизненного цикла схемы документируют, как показано на рис. 177, и преобразуют в формат XML.

Ниже приведен фрагмент текста на языке XML.

```
<Executable>
    <Program_structure>
        <Workplan>
            <Its_elements/>
        </Workplan>
        <Parallel>
            <Branches/>
            <Branches/>
```

element **Executable**

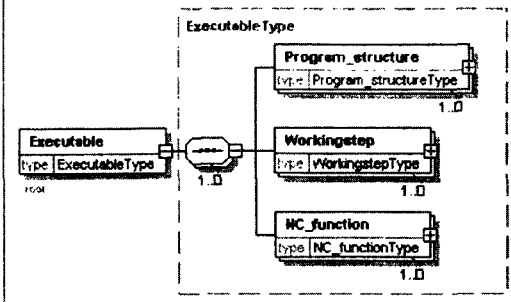
diagram	
type	ExecutableType
children	Program_structure Workingstep NC_function
annotation	documentation root
source	<xs:element name="Executable" type="ExecutableType" abstract="1"> <xs:annotation> <xs:documentation>root</xs:documentation> </xs:annotation> </xs:element>

Рис. 177. Документ исполняемого блока executable

```
            </Parallel>
            <Non_sequent>
                  <Its_elements/>
                  <Its_elements/>
            </Non_sequent>
            <Selective>
                  <Its_elements/>
                  <Its_elements/>
            </Selective>
            <If_statement>
                  <Condition/>
                  <True_branch/>
            </If_statement>
            <Assignment>
                  <Its_lvalue/>
                  <Its_rvalue/>
            </Assignment>
            <While_statement>
                  <Condition/>
                  <Body/>
            </While_statement>
      </Program_structure>
      <Workingstep>
```

```
            <Touch_probe>
                  <Start_position/>
                  <Its_worfkpiece/>
                  <Direction/>
                  <Expected_value/>
                  <Its_probe/>
            </Touch_probe>
            <Rap_movement/>
            <Mach_workingstep>
                  <Its_feature/>
                  <Its_operation/>
                  <Its_effect/>
            </Mach_workingstep>
      </Workingstep>
      <NC_function>
            <Display_mssg/>
            <Optional_stop/>
            <Program_stop/>
            <Set_mark/>
            <Wait_for_mark/>
      </NC_function>
</Executable>
```

Заключение

Жизненный цикл управляющей программы в стандарте ISO 14649 STEP-NC состоит из двух фаз. В первой фазе разрабатывают внешнее представление управляющей программы в стандартах .xsd, .xsl, .xml, во второй фазе создают внутреннюю структуру хранения данных в DOM-форме. Первая фаза использует EXPRESS-модель управляющей программы и инструментальную систему XML Spy. Вторая фаза служит основой для изменений в архитектуре системы ЧПУ.

Список литературы

1. *Сосонкин В.Л., Мартинов Г.М.* Концепция числового программного управления мехатронными системами: архитектура систем типа PCNC // Мехатроника. 2000. №1. С. 26–29.

2. *Мартинов Г.М., Сосонкин В.Л.* Концепция числового программного управления мехатронными системами: реализация геометрической задачи // Мехатроника. 2001. №1. С. 9–15.

3. *Сосонкин В.Л., Мартинов Г.М.* Концепция числового программного управления мехатронными системами: реализация логической задачи // Мехатроника. 2001. №2. С. 3–5.

4. *Мартинов Г.М. Сосонкин В.Л.* Концепция числового программного управления мехатронными системами: проблема реального времени // Мехатроника. 2000. №3. С. 37–40.

5. *Сосонкин В.Л., Мартинов Г.М.* Концепция систем ЧПУ типа PCNC с открытой архитектурой // СТИН. 1998. №5. С. 7–12.

6. См. *www.opcfoundation.org, www.scada.ru, www.osp.ru/os/1999/04/07.htm*

7. *Эммерих В.* Конструирование распределенных объектов. Методы и средства программирования интероперабельных объектов в архитектурах OMG/CORBA, Microsoft/COM, Java/RMI: Пер. с англ. – М.: Мир, 2002.

8. *Густав О., Джангуидо П.* Цифровые системы автоматизации и управления. – СПб.: Невский диалект, 2001.

9. *Эш Рофэйл, Яссер Шохауд.* COM и COM+. Полное руководство: Пер. с англ. – К.: ВЕК +; К.: НТИ; М: Энтроп, 2000.

10. *Banbury-Masland B.* OPC's success based on teamwork, technology, and process. I&CS. 1997. January. P. 78–79.

11. OPC software speeds up process of connecting OPC client systems to OPC sever. I&CS. 1999. August. P. 28.

12. *http://www.steptools.com.*

13. *http://www.okstate.edu/ind-engr/step/WEBFILES/Papers.*

14. *http://www.cals.ru/structstep.html#str.*

15. *http://www.cals.ru/structstep.html#ex.*

16. *http://www.xml.coverpage.org/StepExpressXML.html#step-part28.*

17. Microsoft System Journal, Q&A. 1996. April. P. 89–101.

18. Microsoft COM Specification, version 0,9, 10/24/95 (Available from Microsoft FTP site).

19. OLE Automation Programming Reference, Microsoft Press, Redmond, WA, 1996.

20. *Сосонкин В.Л.* Задачи числового программного управления и их архитектурная Реализация // Станки и инструмент. 1988. №10. С. 39–40.

21. *Сосонкин В.Л., Мартинов Г.М.* Принципы построения систем ЧПУ с открытой Архитектурой // Приборы и системы управления. 1996. №8. С. 18–21.

22. *Сосонкин В.Л., Мартинов Г.М.* Современное представление об архитектуре систем ЧПУ типа PCNC // Автоматизация проектирования. 1998, №3(9). С.35–39.

23. *Жданов А.* Операционные системы реального времени // PC WEEK / RE. 1998. № 8(182). С. 17–18; №9 (183). С.24.

24. *Хухлаев Е.* Операционные системы реального времени и Windows NT // Открытые системы. 1997. № 5. С. 48–51.

25. *Сосонкин В.Л.* Разработка диспетчеров для систем управления с персональным компьютером // Приборы и системы управления. 1995. №2. С.14–18.

26. *Labs W.* MS Windows: In control? // I&CS. 1998. July. P. 48–56, 59.

27. *Dexter M., Kok R.* Windows NT goes embedded // I&CS. 1999. July. P. 75–76.

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

29. *J. Stenerson.* Fundamentals of Programmable Logic Controllers, Sensors and Communications. – Prentice Hall. 1998. P. 562

30. *G. Dunning.* Introduction to Programmable logic Controllers. – Delmar Publishers. 1998. P. 433.

31. *Теркель Д.* OLE for Process Control – свобода выбора // Современные технологии автоматизации. 1999. №3. С. 28–32.

32. *Дейл Роджерсон* Основы COM: Пер. с англ. – М.: Русская Редакция ТОО «Channel Trading Ltd.», 1997.

33. *Мартинов Г.М.* Открытая система ЧПУ на базе общей магистрали // Автомобильная промышленность. 1997. №4. С. 31–34.

34. *Джон Пьюполо* OLE: создание элементов управления: Пер. с англ. – К.: Издательская группа BHV, 1997.

35. *Вебер Д.* Технология Java в подлиннике: Пер. с англ. – СПб.: БВХ-Петербург, 2000.

36. Технология «тонкий клиент/сервер» компании Citrix. http://www.deltacomm.ru/vendors/products/citrix/client.htm

37. *Кватрани Т.* Rational Rose 2000 и UML. Визуальное моделирование: Пер. с англ. – М.: ДМК Пресс, 2001. (Серия «Объектно-ориентированные технологии в программировании»).

38. *Сосонкин В.Л., Мартинов Г.М.* Концепция числового программного управления мехатронными системами: построение межмодульной коммуникационной среды // Мехатроника. 2000. №6. С. 2–7.

39. *Мартинов Г.М., Сосонкин В.Л.* Концепция числового программного управления мехатронными системами: реализация терминальной задачи // Мехатроника. 2001. №4. С. 2–8.

40. *Сосонкин В.Л., Мартинов Г.М.* Концепция числового программного управления мехатронными системами: интеграция на основе комплекса производственных стандартов STEP (Standard for the Exchange of Product model data) // Информационные технологии в проектировании и производстве. – М.: ВИМИ, 2003. №2. С. 38–44.

41. *Weck M., Wolf J.* Introduction to STEP-NC, A standard providing data for modern NC machining enabling enhanced functionality // Laboratory for Machine Tools and Production Engineering Aachen University of Technology. Aachen. 2003. February 12. P. 52.

42. *Yonk Tak Hyun.* Know-how feedback based on manufacturing features (STEP-NC Server) // Laboratory for Machine Tools and Production Engineering Aachen University of Technology. Aachen. 2003. February 12. P. 25.

43. *Cho J.H., Sub S.H.* On-line tool path generation and modification for STEP-NC. // Journal of CAD/CAM. 1999. Vol. 4. №4. P. 295–311.

44. *Newman S.T., Allen R.D., Ross R.S.U.* CAD/CAM solution for STEP compliant CNC Manufacture. // Proceedings of the 1st CIRP (UK) Seminar on Digital Enterprise Technology. – University of Durham, 2000. P. 124–128.

45. ISO 14649. Data model for Computerized Numerical Controllers. Part 10: General Process Data. Draft International Standard ISO/FDIS 14649-10, version 4. 2001. August. P. 180.

46. *Muller P.* NC-Controllers and STEP-NC + Status of Standardization ISO 14649. // Hanover: EMO 12–19. 2001. №9. P.8. (See www.siemens.de).

47. *Hardwick M.* e-manufacturing using STEP-NC. (See www.steptools.com).

48. *Hardwick M.* Tools for Implementing STEP&STEP-NC. (See www.steptools.com).

49. European STEP-NC Project. (See www.step-nc.org).

50. *Мартинов Г.М., Сосонкин В.Л.* Концепция числового программного управления мехатронными системами: структура руководства по программированию // Мехатроника. 2001. №8. С. 20–27.

51. *Сосонкин В.Л., Мартинов Г.М.* Концепция геометрического ISO-процессора для систем ЧПУ // СТИН, 1994. №7. С. 12–20.

52. *Соломенцев Ю.М., Сосонкин В.Л., Мартинов Г.М.* Построение персональных систем ЧПУ (PCNC) по принципу открытых систем // Информационные технологии и вычислительные системы, 1997. №3. С. 68–75.

53. *Перцовский М.* Время распределять системы и время собирать системы // PC WEEK/RE, 1997. № 37 (111). С. 57–61.

54. *Otto H.-P., Rath G.* State diagrams. A new programming method for Programmable Logic Controllers / Software Engineering for Manufacturing Systems: Methods and CASE tools / Edited by A.Storr and D.Jarvis. – London: Chapman&Hall, 1996. P. 27–37.

55. *Сосонкин В.Л., Мартинов Г.М., Любимов А.Б.* Интерпретация диалога в windows-интерфейсе систем управления // Приборы и системы управления. 1998. №12. С. 10–13.

56. *B. Selic, G. Gullekson, P. Ward* Real-Time Object-Oriented Modelling. Computer Society. 1994. P. 560.

57. *http://www.can-cia.de.*

58. *Буч Г., Рамбо Д., Джекобсон А.* Язык UML. Руководство пользователя: Пер. с англ. – М.: ДМК Пресс, 2001. (Серия «Для программистов»).

59. *Сосонкин В.Л.* Программное управление технологическим оборудованием. Учебник для вузов. – М.: Машиностроение, 1991.

60. *Сосонкин В.Л., Мартинов Г.М.* Принципы построения систем ЧПУ с открытой архитектурой // Приборы и системы управления. 1996. №8. С. 18–21.

61. *Сосонкин В.Л., Мартинов Г.М.* «AdvancEd» – универсальная среда для редактирования, отладки и моделирования программ ЧПУ в коде ISO-7bit (любой версии) //Автотракторное электрооборудование. 2001. №1–2. С. 41–42.

62. *Грис Д.* Конструирование компиляторов для цифровых вычислительных машин. Пер. с англ. – М.: Мир. 1975. 544 с.

63. *Сосонкин В.Л., Тилеш Ю.* Представление о процессорном устройстве числового программного управления оборудованием как виртуальном вычислителе // Машиноведение. 1981. №6. С. 50–57.

64. *Оберг Р. Дж.* Технология COM+. Основы и программирование.: Пер. с англ. Учеб. пособие – М.: Издательский дом "Вильямс", 2000.

65. *Мартинов Г.М.* Виртуальные приборы диагностики в системе ЧПУ // Информатика-машиностроение. 1998. №4. С. 8–12.

66. *Деннинг А.* ActiveX для профессионалов. – СПб.: Питер, 1998.

67. *Сосонкин В.Л., Мартинов Г.М.* Концепция числового программного управления мехатронными системами: реализация диагностической задачи управления // Мехатроника. 2001. №3. С. 2–6.

68. *B. Meyer* Object-Oriented Software Construction. Computer Society. 1997. April. P. 1250.

69. *D. Wright, D. Williams* Object – oriented software design techniques for process control / Trns Inst MC. 1994. Vol. 16. №1. P. 48–56.

70. *Буч Г.* Объектно-ориентированное проектирование с примерами приложений на C++. 2-е изд.: Пер. с англ. – М.: Бином; СПб: Невский диалект, 1998.

71. *Фаулер М., Скот К.* UML в кратком изложение. Применение стандартного языка объектного моделирования: Пер. с англ. – М.: Мир, 1999.

72. *Сосонкин В.Л.* Принципы построения персональных систем ЧПУ с открытой архитектурой // Труды междун. конф. «Информационные средства и технологии, 21–23 окт. 1997». М.: Междунар. академия информатизации. 1997. С. 154–159.

73. *Мартинова Л.И., Мартинов Г.М.* Практические аспекты реализации модулей открытой системы ЧПУ // Автотракторное электрооборудование. 2002. №3. С. 31–37.

74. *Липаев В.В.* Документирование и управление конфигурацией программных средств. Методы и стандарты. Сер. «Информатизация России на пороге XXI века». – М.: Синтег, 1998.

75. *Нортон П.* Персональный компьютер фирмы IBM и операционная система MS-DOS: Пер. с англ.– М.: Радио и связь, 1991.

76. *Лещенко В.А., Богданов Н.А., Вайнштейн И.В. и др.* Станки с числовым программным управлением (специализированные). – М.: Машиностроение, 1988.

77. *Мюллер Дж.* Visual Studio 6. Полное руководство: Пер. с англ. – К.: Издательская группа BHV, 1999.

78. *Кумсков М.* UML, Rose98i – далее везде // PC WEEK /RE. 1999. № 29–30 (203–204). С .26–27.

79. *Новорусский В.В.* Конечно-автоматные системы управления (принципы построения и анализ поведения). – Новосибирск: Наука, 1982.

80. *Storr A., Jarvis D.* Software Engineering for Manufacturing Systems. Methods and CASE tools. – Chapman & Hall, London, Weinheim, New York, Melbourne, Madras, 1996. P. 199.

81. *Оберг Р. Дж.* Технология COM+. Основы и программирование. : Пер. с англ. Учеб. пособие. – М. : Издательский дом «Вильямс», 2000.

82. *Мартинов Г.М., Сосонкин В.Л.* Концепция числового программного управления мехатронными системами: технология объектно-ориентированного программирования // Мехатроника. 2001. №7. С. 5–9.

83. *Трельсен Э.* Модель COM и применение ATL 3.0: Пер. с англ. – СПб.: Издательская группа BHV, 2000.

84. *Сосонкин В.Л., Мартинов Г.М.* Принципы построения систем ЧПУ с открытой архитектурой // Приборы и системы управления. 1996. №8. С. 18–21.

85. *Сосонкин В.Л.* Некоторые принципы разработки систем ЧПУ нового поколения // СТИН. 2000. №9. С. 24–29.

86. *Сосонкин В.Л., Мартинов Г.М.* Понятийный аппарат комплекса производственных стандартов для числового программного управления оборудованием, – ISO 14649 STEP-NC (Standard for the Exchange of Product model data for NC) // Мехатроника, автоматизация, управление. 2004. №8. С. 37–44. (См. www.ncsystems.ru).

87. *Мартинов Г.М., Сосонкин В.Л.* Формализация данных STEP-NC-формата: фаза построения UML-модели // Мехатроника, автоматизация, управление. 2005. №1. С. 49–56.

88. XML Spy. User and Reference Manual (See www.altova.com).

89. *Galloway T.* Principles of XML Schema design. (See www.RenderX.com).

Учебное издание

Владимир Лазаревич Сосонкин
Георгий Мартинов Мартинов

СИСТЕМЫ ЧИСЛОВОГО ПРОГРАММНОГО УПРАВЛЕНИЯ

Редактор *Е.В. Комарова*
Корректор *Л.И. Трифонова*
Оформление *Т.Ю. Хрычевой*
Компьютерная верстка *С.Н. Яковлевой*

Подписано в печать 10.08.2005. № 78(и). Формат 60x90/16
Печать офсетная. Бумага офсетная. Печ. л. 18,5.
Тираж 2000. Заказ № 2225.

Издательская группа «Логос»
105318, Москва, Измайловское ш., 4

Отпечатано с готовых диапозитивов в типографии
ФГУП «Издательство «Самарский Дом печати».
443080, г. Самара, пр. К. Маркса, 201.
Качество печати соответствует качеству предоставленных диапозитивов