Модели математического обеспечения открытых систем ЧПУ

Существует достаточно единодушное мнение разработчиков систем ЧПУ, станкостроителей и конечных пользователей, что архитектура систем должна быть открытой. За последние годы было выполнено несколько международных проектов, цель которых состояла в создании модели открытой системы ЧПУ. Результаты проектов можно посчитать достаточно успешными хотя бы потому, что многочисленные производители эти результаты практически приняли. Между тем, робкие попытки создать современную отечественную систему ЧПУ свидетельствуют о том, что международный опыт игнорируется. Цель последующего изложения состоит в том, чтобы обратить внимание на ряд интересных решений, которыми можно было бы воспользоваться в практике разработки математического обеспечения ЧПУ. Другая цель заключается в информировании заинтересованной инженерной аудитории относительно реальной архитектуры систем ЧПУ, которые поступают в нашу страну по импорту

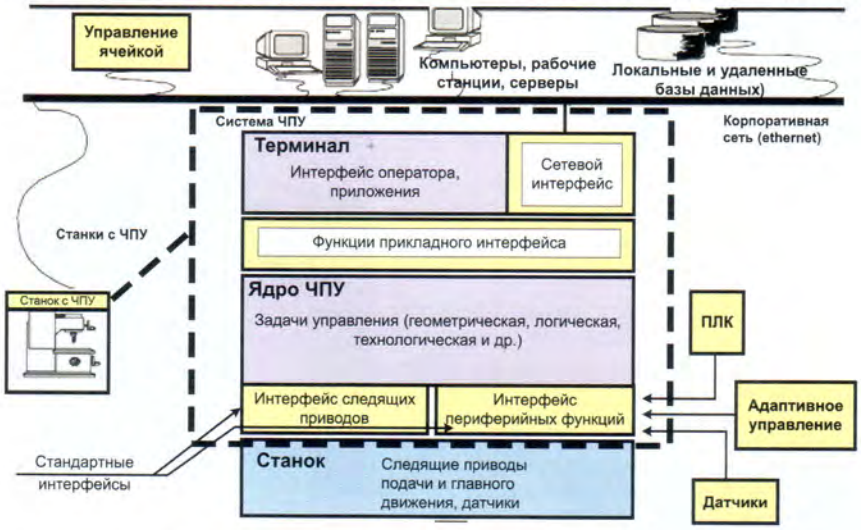


Рис. 1. Виртуальная машина ЧПУ. ПЛК – программируемый логический контроллер.

Модели открытой системы ЧПУ

Разработка модели означает создание концепции. Система ЧПУ слишком сложна, чтобы можно было ограничиться одной моделью, и обычно рассматривают их иерархическое вложение. Наиболее общей моделью системы ЧПУ служит многоуровневая виртуальная машина, пример которой показан на рис. 1.

В этой модели представлены: уровень терминала (интерфейс оператора, приложения) с машинным масштабом работы всех его компонентов; уровень задач управления, работающих в реальном времени; уровень объектов управления на станке (следящие приводы, электроавтоматика).

Каждому уровню такой машины могут быть сопоставлены свои модели. Достаточно полный комплекс таких моделей перечислен на рис. 2.

В сравнении с виртуальной машиной здесь разделены платформа и прикладная часть математического обеспечения, а в прикладной части обозначены модели пользователя, модели приложений и модели услуг. Усилия участников международных проектов были сосредоточены вокруг создания моделей прикладной части математического обеспечения.

Модели проекта OSACA. В архитектуре проекта *OSACA (Open Systems Architecture Controls within Automation Systems*, архитектура открытых систем управления для автоматизации) был сделан акцент на разработку коммуникационной среды, поддерживающей взаимодействие прикладных модулей между собой и с системной платформой [1], [2].

Практика показывает, что как-то стандартизировать можно лишь внешнее поведение прикладных модулей, поскольку заложенное в них технологическое «know-how» необходимо защищать; а разработчикам математического обеспечения должна быть предоставлена свобода программирования модулей в их собственной манере. Поэтому усилия авторов проекта были сосредоточены вокруг тщательного специфицирования прикладных интерфейсов *API (Application Programming Interface)* на стыке приложений и их инфраструктуры. Возможность использования практически любой платформы позволяет *OSACA-приложениям* выстраивать систему управления с использованием одного компьютера или разных компьютерных систем, см. рис. 3.

В системную платформу в *OSACA-архитектуре* интегрированы операционная система, коммуникационная среда и средства конфигурации, которые используются для построения топологии математического обеспечения из доступных модулей с целью достижения заданной функциональности. Доступ к системной платформе осуществляется через *API*. Интерфейс *API* должен быть нейтральным,

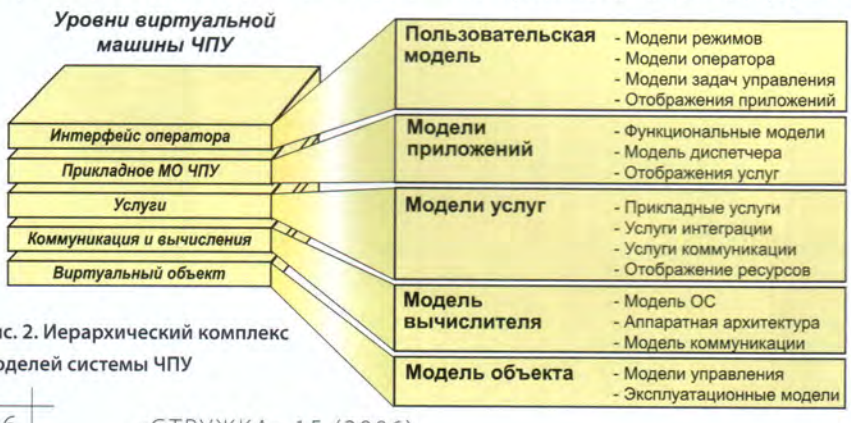
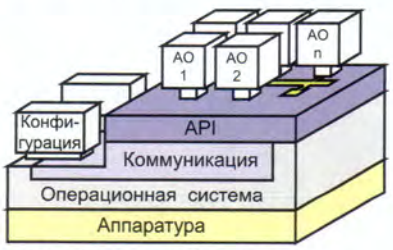


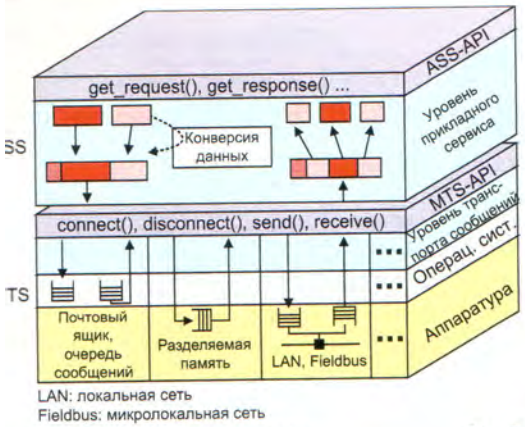
Рис. 2. Иерархический комплекс моделей системы ЧПУ



АО: архитектурный объект
API: прикладной интерфейс

Рис. 3. Архитектура OSACA. АО: ***Architectural Object, архитектурный объект; HMI: Human Machine Interface, интерфейс оператора; MCAO: Master configuration AO, ведущий конфигурационный архитектурный объект; SCAO: Slave Configuration AO, ведомый конфигурационный архитектурный объект***

«СТРУЖКА» • e-mail: editors@struzhka.ru



ис. 4. Коммуникационная среда OSACA-архитектуре

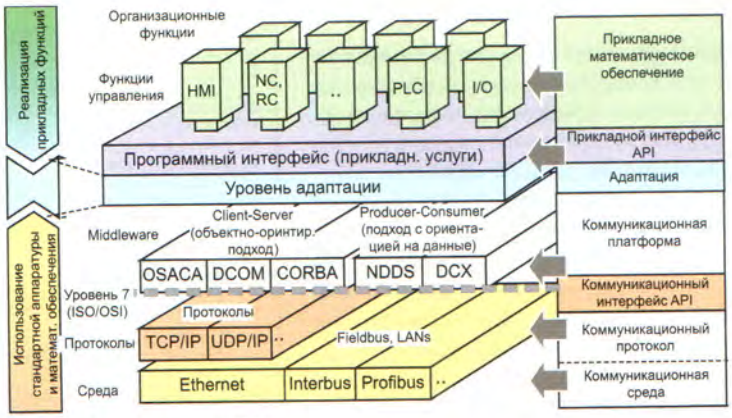


Рис. 5. Детализация структуры уровня MTS.
HMI: Human-Machine-Interface, интерфейс оператора; NC: Numerical Control, система ЧПУ станка; RC: Robot Control, система ЧПУ робота; PLC: Programmable Logic Control, контроллер электроавтоматики; I/O: Input/Output, входы-выходы; Middleware: промежуточное программное обеспечение; DCOM: Distributed Component Object Model, распределенная модель компонентных объектов; CORBA: Common Object Request Broker Architecture, архитектура брокеров объектных запросов; TCP/IP: Internet-протокол; UDP: User Datagram Protocol, дэйтаграммный протокол; NDDS: Network Data Delivery Service, сетевая служба доставки сервиса (компания Real-Time Innovations, Inc); DCX: Data eXchange, протокол обмена данными.

тобы использовать переносимые модули, том числе и от разных разработчиков.

В силу объектно-ориентированного остроения математического обеспе-ения прикладные модули получили аименование архитектурных объектов *AO (Architectural Objects)*. Единственным средством информационного обмена ежду архитектурными объектами, как в ределах одной вычислительной среды, ак и за ее пределами в распределен-

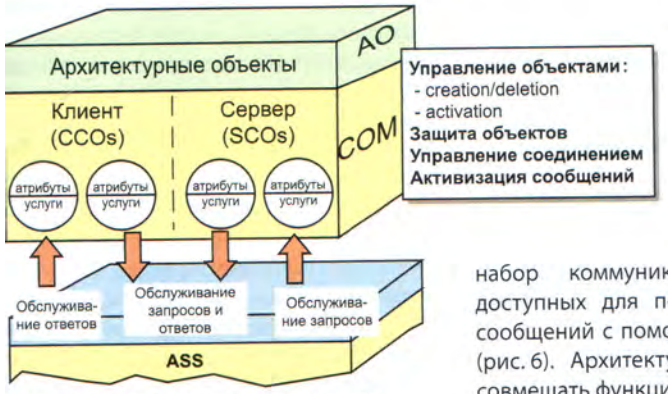


Рис. 6. Интерфейс коммуникационного объекта, совмещающий функции клиента и сервера

ной системе, служит коммуникационная среда. Стандартные протоколы коммуникационной среды обеспечивают единообразные форматы данных и фиксированные наборы сообщений. Система протоколов производна по отношению к базовой архитектуре открытых систем *OSI*. Она представлена двумя крупными уровнями (рис. 4): уровнем транспорта сообщений *MTS (Message Transport System)*, эквивалентным четырем нижним слоям 1-4 архитектуры *OSI*; а также уровнем прикладного сервиса *ASS (Application Services System)*, эквивалентным трем верхним слоям 5–7 архитектуры *OSI*.

Уровень *MTS* предлагает сервис транспорта произвольных сообщений, с предварительной установкой соединения, между архитектурными объектами *AO*. Этот уровень может быть адаптирован к любым существующим механизмам обме-

на информацией (рис. 5).

Уровень *ASS* имеет дело с прикладным протоколом, выстроенным на основе клиент-серверных отношений с использованием объектно-ориентированного подхода. В серверном архитектурном объекте любая информация, данные или услуги, которые доступны извне, привязаны к коммуникационному объекту *CO (Communication Object)*. С точки зрения клиента, сервер представляет собой набор коммуникационных объектов, доступных для передачи и получения сообщений с помощью услуг уровня *ASS* (рис. 6). Архитектурные объекты могут совмещать функции клиента и сервера.

Существует фиксированный набор классов коммуникационных объектов, наиболее важными из которых являются класс переменной (*variable*) для записи и чтения данных и класс процесса (*process*) для переключения состояний в конечных автоматах. Дополнительный класс события (*event*) служит для передачи инициативных событий и сообщений. Эти коммуникационные объекты создаются в каждом архитектурном объекте и регистрируются в менеджере коммуникационных объектов *COM (Communication Object Manager)*, который взаимодействует с уровнем *ASS*. Один серверный коммуникационный объект *SCO* одновременно доступен различным клиентским коммуникационным объектам *CCO*.

При разделении транспортных и прикладных услуг разработчик математического обеспечения ЧПУ может полностью сосредоточиться на решении специфических задач управления. Чтобы предложить некоторую функциональность одного модуля для внешнего доступа другим модулям, необходимо определить набор соответствующих коммуникационных объектов и установить необходимые связи с уровнем прикладного сервиса (рис. 7).

Определены два типа интерфейсов: интерфейс данных и интерфейс процессов, см. рис. 8. Интерфейс данных служит для чтения данных архитектурного объекта и записи данных в архитектурный объект. Доступ осуществляется посредством коммуникационного объекта класса «variable». Каждая внутренняя переменная, представленная в архитектурном

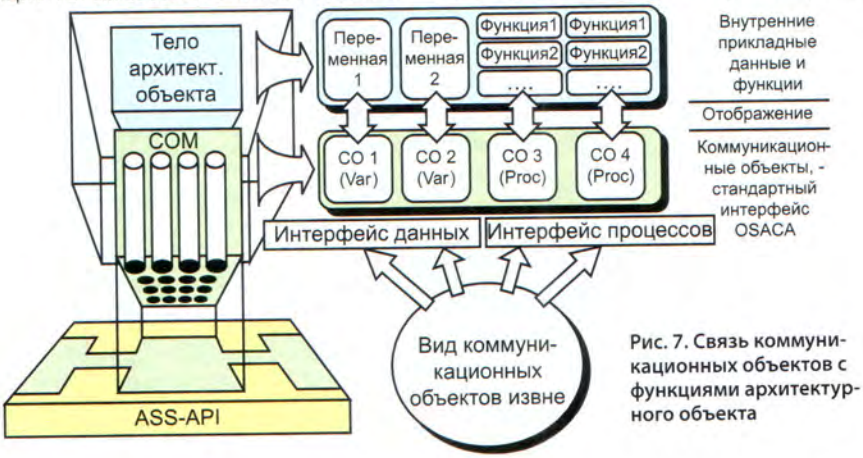


Рис. 7. Связь коммуникационных объектов с функциями архитектурного объекта

объекте физически и имеющая внешний доступ, должна иметь свое отображение в реализации коммуникационного класса. Пара «внутренняя переменная – коммуникационный объект» служит атрибутом архитектурного объекта. Атрибуты доступны извне с помощью услуг *«get»* и *«set»* коммуникационной среды.

Интерфейс процессов определяет динамическое поведение архитектурного объекта. Переключение его состояний путем управления соответствующим конечным автоматом позволяет выбрать доступную функциональность.

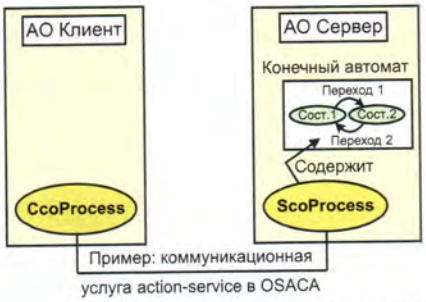


Рис. 8. Работа интерфейсов данных и процессов архитектурного объекта

Существуют пять типов архитектурных объектов: интерфейс оператора, или терминальная задача, *MMC (Man Machine Control)*; управление автоматикой, или логическая задача, *LC (Logic Control)*; ядро ЧПУ, или геометрическая задача, *MC (Motion Control)*; управление следящими приводами *AC (Axis Control)*; управление рабочим процессом, или технологическая задача, *PC (Process Control)*. Рис. 9 представляет обзор некоторых коммуникационных объектов, определенных для архитектурного объекта «Ядро ЧПУ».

На рис. 10 представлена архитектура математического обеспечения, выстроенная согласно описанному сценарию, совместно с трех-координатным фрезерным станком. Использована операционная система *VxWorks*, а коммуникационный уровень *OSACA* выстроен над *TCP/IP*.

Коммуникационная среда с использованием архитектуры *CORBA*. В сложных

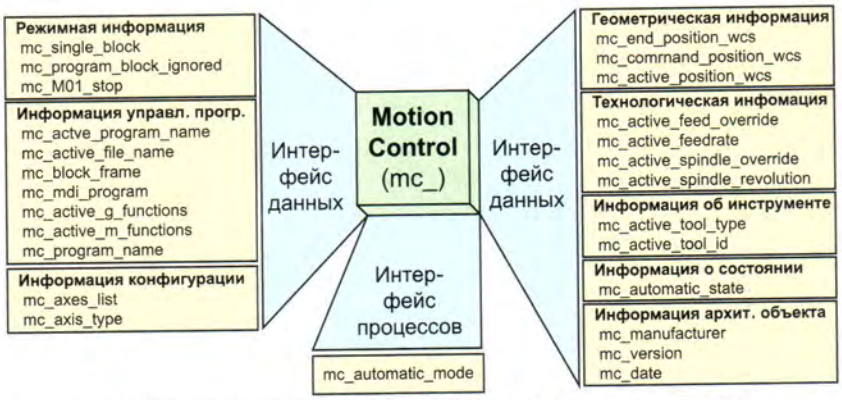


Рис. 9. Имена коммуникационных объектов в интерфейсе ядра системы ЧПУ

распределенных объектных программных системах управления достаточно широкое распространение получила «архитектура брокеров объектных запросов» *CORBA (Common Object Request Broker Architecture)*, [3]. Простейшие схемы взаимодействия брокеров *ORB* показаны на рис. 11.

Важнейшими понятиями, введенными на рис. 11, являются *IDL*, *Stub* и *Skeleton*.

Язык описания интерфейсов *IDL* используют в качестве средства доступа к удаленным сервисам. Цель создания языка состоит в необходимости описания интерфейсов объектов, независимо от языка реализации самого объекта. При разработке конкретных приложений на базе *ORB*, *IDL-описания* интерфейсов используемых объектов транслируются в наборы функций доступа к ORB, которые затем связываются с исполняемым модулем. *Stub* – набор функций на языке программы-клиента, который сгенерирован из *IDL*-описания интерфейса объекта-клиента; *Skeleton* – набор функций, который сгенерирован на основе *IDL*-описания интерфейса объекта-сервера, используемый *ORB* для вызова метода объекта-сервера, запрошенного объектом-клиентом. На стороне клиента *объект-stub* играет роль локального представителя удаленного объекта. В коде, генерируемом в *stub*, заложены знания о том, что требуется сделать для обращения к методу удаленного объекта. На стороне сервера *объект-skeleton* служит посредником при доступе к удаленному объекту по правилам его системы программирования, при этом *skeleton* распознает удаленность обращения. Объектная реализация (*Servant*) выполняет операции, сформулированные в *IDL-интерфейсе*; *servant* может представлять один или несколько объектов.

Для доставки сообщения от клиента к серверу и получения ответных результатов необходим системный компонент, отвечающий за выполнение подобных функций. В архитектуре *CORBA* такой компонент называется брокером объектных запросов *ORB (Object Request Broker)*, в функции которого, в том числе, входит передача данных в машинно-независимом формате от клиента серверу и от сервера клиенту. Кроме того, *ORB* отвечает за правильное указание сетевого адреса объекта-сервера. В архитектуре *CORBA* каждой объектной реализации сопоставлена уникальная объектная ссылка *object reference*, которая используется клиентом для указания брокеру *ORB*.

Взаимодействие брокеров *ORB* архитектуры *CORBA* поддерживается «универсальным меж-брокерным протоколом» *GIOP (General Inter-ORB Protocol)*. Универ-

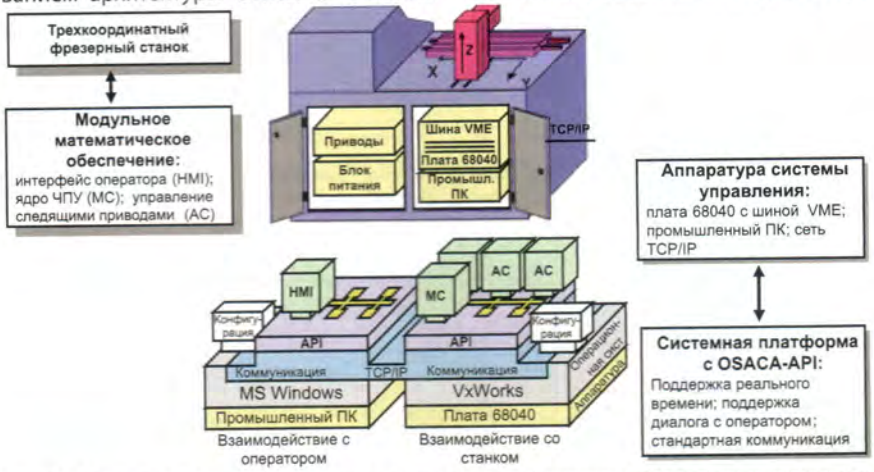


Рис. 10. Математическое обеспечение распределенной системы ЧПУ трех-координатного фрезерного станка. ПК: персональный компьютер.

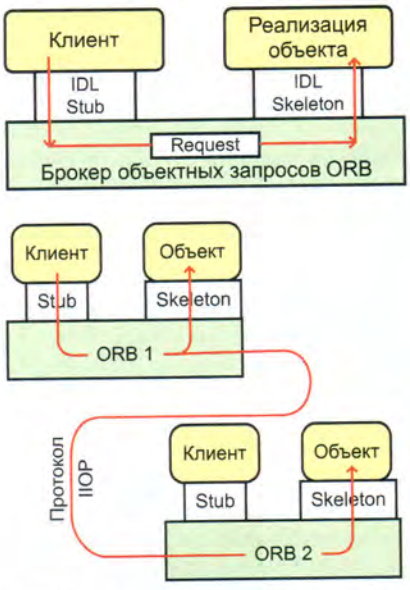


Рис. 11. Простейшие схемы взаимодействия брокеров ORB. IDL – Interface Definition Language, язык определения интерфейса; Stub – «заглушка»; Skeleton – «заготовка».

«СТРУЖКА» • e-mail: editors@struzhka.ru

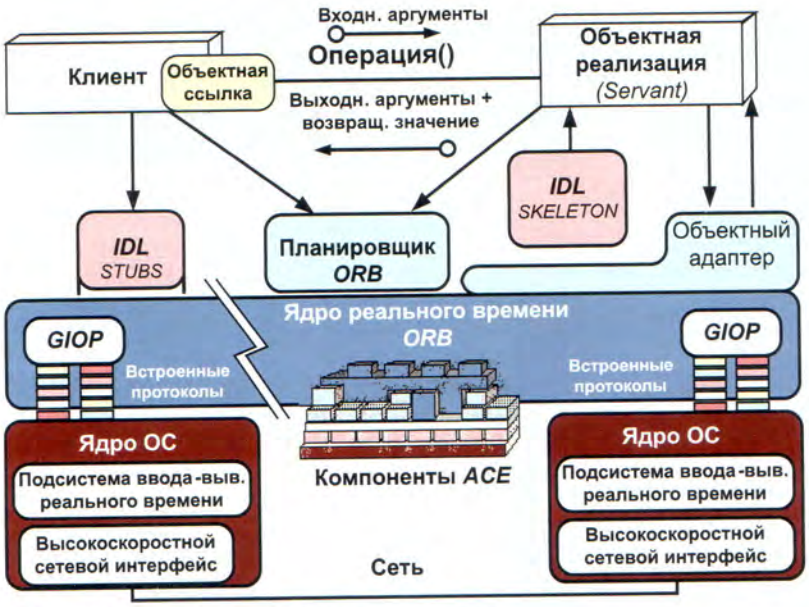


Рис. 12. Взаимодействие брокеров в реальном времени

сальность протокола состоит в том, что он не зависит от конкретной сетевой транспортной среды и может быть отображен на любой транспортный протокол, поддерживающий виртуальные соединения. Так, отображение *GIOP* на протокол *TCP/IP* называют «меж-брокерным *Internet-протоколом*» *IIOP (Internet Inter-ORB Protocol)*. Назначение протоколов *GIOP/IIOP* заключается в том, чтобы поддержать сети брокеров в рамках или вне *Internet*.

Клиент-серверная структура взаимодействия брокеров *ORB* в реальном времени показана на рис. 12.

Некоторые модули этой структуры требуют дополнительного пояснения. Адаптивная коммуникационная среда *ACE (Adaptive Communication Environment)* представляет собой объектно-ориентированную структуру с открытыми кодами, которая располагает коммуникационными шаблонами, [4]. Шаблоны и компоненты, разработанные в среде *ACE*, привели к созданию коммуникационной среды для брокеров *ORB*, получившей наименование *TAO (The ACE ORB)*. Система *TAO* представляет собой «промежуточное» (*middleware*) математическое обеспечение для *CORBA*, которое позволяет клиентам вызывать операции в распределенных объектах, не заботясь об их расположении, языке программирования, операционной среде, коммуникационных протоколах, внутренних связях и аппаратной поддержке [5], [6].

На рис. 13 показано, как компоненты *TAO* структурированы и оптимизированы для повышения быстродействия и масштабируемости, [7]. Коммуникационная среда представляет собой многоуровневую среду, траектории сообщений через которую могут различаться при «*collocated*» и «*uncollocated*» взаимодействиях.

Среда *TAO* предлагает два типа коммуникации между компонентами: для «collocated» компонентов, размещенных в едином адресном пространстве; для «*uncollocated*» компонентов (локальных или удаленных), размещенных в разных адресных пространствах. В первом случае существует возможность сократить коммуникационный путь за счет обхода уровней маршалинга, маршрутизации, демультиплексирования и др., что, конечно же, существенно повышает быстродействие. Во втором случае есть другая возможность повышения быстродействия на основе использования модификации сетевой подсистемы *Linux* под названием *RTnet*. *RTnet* предоставляет собой сеть жесткого реального времени поверх стандартного протокола *IP* (сеть разработана компанией *Embedix Inc.*, [8], известной как *Lineo*). Сеть работает с ядром *Linux 2.2.x*, с расширением *RTLinux 2.3* или прикладным интерфейсом реального времени *RTAI 1.3 (Real Time Application Interface)*.

В системах управления пространства задач реального времени и пользовательских приложений обычно обособлены и требуют межпространственной коммуникации с помощью быстрого и надежного транспорта. С этой целью разработан встроенный TAO-протокол с использованием разделяемой памяти *SHMIOP (Shared Memory Inter-ORB Protocol)*, использующий *RTAI*, [9].

Чтобы распределенная система управления работала в реальном времени, ее компоненты должны взаимодействовать через коммуникационную среду по различным траекториям. На рис. 14 показана схема интерактивности компонентов системы ЧПУ из проекта *OCEAN (Open Controller Enabled by an Advanced real-time Network)*, [10]. Цель проекта состояла в том, чтобы построить открытую систему ЧПУ, в которой могли бы быть использованы встроенные компоненты от различных разработчиков. Проект *OCEAN* унаследовал многие идеи *OSACA*, однако использовал операционную систему

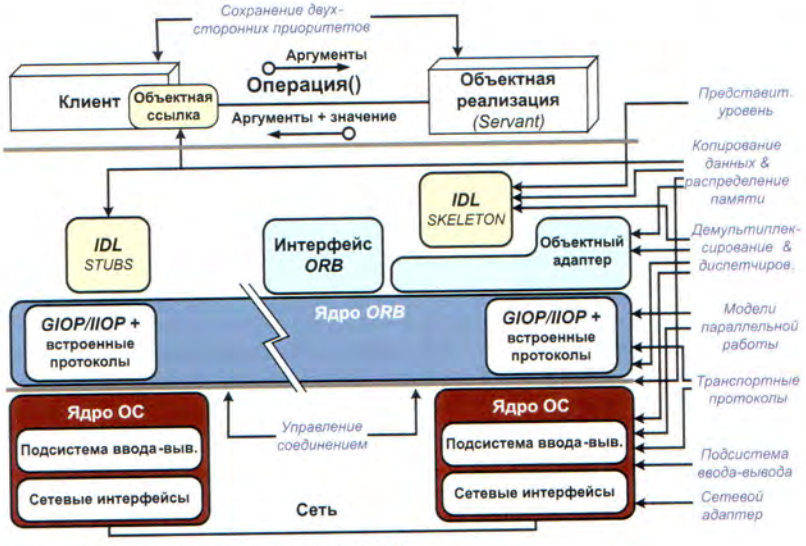


Рис. 13. Многоуровневая архитектура TAO

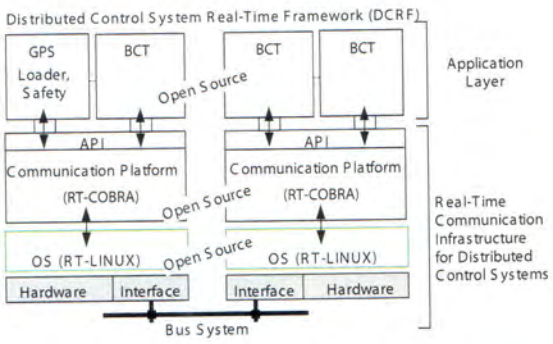


Рис. 14. Архитектурная платформа DCRF (Distributed Control System Real-Time Framework) из проекта OCEAN. LXRT: Linux с расширением RTLinux

Linux и стандартные коммуникационные средства *CORBA*.

Объединение моделей *CORBA* и *MMS*. Интересное решение представлено в [11], где предлагается объединить *CORBA* с объектно-ориентированным представлением *MMS* (ISO 1990).

Спецификация производственных сообщений *MMS (Manufacturing Message Specification)* представляет собой прикладной уровень протокольного стека ISO/OSI. Функции уровня позволяют организовать удаленное управление и мониторинг для таких компьютеризованных объектов, как станки, роботы и другие автоматические устройства. С позиций *MMS* компьютеризованные объекты представляют собой виртуальные производственные устройства *VMD (Virtual Manufacturing Devices)*, которые оказывают услуги удаленным пользователям (*Users*) или клиентам. Коммуникация выстроена на базе интерактивных транзакций (*Association*) и реализуется посредством обменных

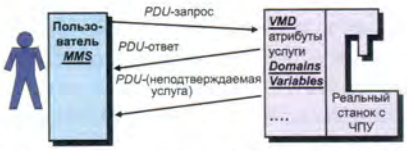


Рис. 15. Удаленное управление станком на основе протокола MMS

сообщений *PDU (Protocol Data Units)* стандартного формата (рис. 15). Коммуникация по большей части носит характер подтверждаемых услуг (*positive, negative, error*), запрошенных удаленным пользователем. Существует также небольшой набор неподтверждаемых услуг в виде сообщений сервера (например, уведомление об изменении статуса *VMD*).

Виртуальное устройство *VMD* служит абстракцией реальной удаленной системы. Это устройство представляет собой объект, который инкапсулирует атрибуты (*identity, status,...*), а также методы (*services, услуги*). Для VMD разработан некоторый набор классов объектов, наиболее известные из которых:

- домены (*Domains*), представляющие собой загружаемые наборы ресурсов пользователя (данные и код);
- программные вызовы (*Program Invocations*), представляющие собой исполняемые программы (подпрограммы) пользователя;
- переменные и наборы переменных *(Variables, Sets of variables)*, принадлежащие *VMD* или домену;
- объекты-события *(Event related objects)*.

Шина *CORBA* вполне может быть приспособлена для коммуникации между архитектурными объектами *AO (Architecture Objects*, понятие из проекта *OSACA*). Для этого потребуется служба межобъектной коммуникации *ORB*, работающая в реальном времени.

В качестве примера на рис. 16 рассмотрена распределенная система ЧПУ на базе персонального компьютера. Коммуникация между удаленным терминалом и локальной системой ЧПУ поддержана соответственно стандарту *MMS*. Архитектурные объекты *AO* дополнительно декомпозированы на функциональные блоки: интерфейс оператора *MMI*; диспетчер каналов *MCM*; ядро ЧПУ (интерпретатор-интерполятор) *MC*; контроллер осевого перемещения *AC*; контроллер шпинделя *SC*; контроллер автоматики *LC*.

На рис. 17 показана структура наложения протоколов *MMS* на шину *CORBA*. Такое решение позволяет отделить терминальные части систем ЧПУ от модулей реального времени в распределенной

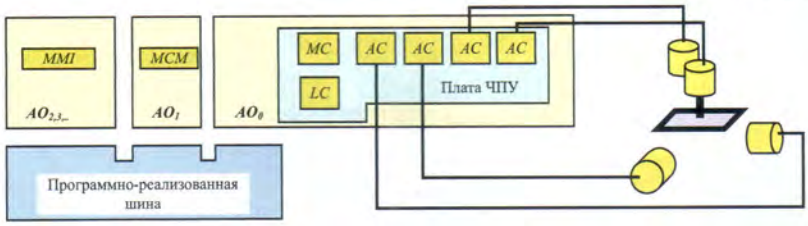


Рис. 16. Архитектура системы ЧПУ. MMI: Man-Machine-Interface, интерфейс оператора; MCM: Motion Control Manager, диспетчер каналов; MC: Motion Controls, ядро ЧПУ (интерпретатор-интерполятор); AC: Axis Controls, контроллер осевого перемещения; SC: Spindle Controls, контроллер шпинделя; LC: Logic Controller, контроллер автоматики; AO: Architectural Object, модуль системы.

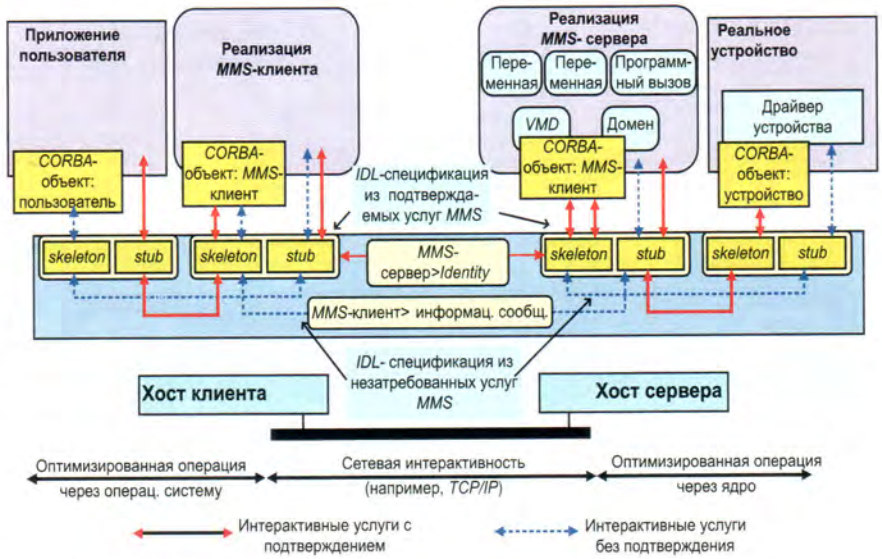


Рис. 17. Принципы наложения MMS на CORBA

системе управления, объединяющей различные системы ЧПУ между собой в единой производственной среде. Возникает возможность использовать несколько терминалов с одним удаленным модулем реального времени или использовать один терминал с несколькими модулями реального времени.

Заключение. Модели, представленные в проектах *OSACA* и *OCEAN*, в стандартах *CORBA* и *MMS* дают достаточно материала для разработки эффективного математического обеспечения открытой системы ЧПУ. Они в особенности полезны при создании системы ЧПУ «с нуля», когда груз предыдущих решений не оказывает давления на разработчиков. Эти модели полезны и в методическом плане, поскольку трудно представить что-нибудь более наглядное для объяснения принципов работы современной системы ЧПУ в целом.

Сосонкин В. Л., профессор, д.т.н.
Мартинов Г. М., профессор. д.т.н.
book@ncsystems.ru

Список литературы.
1. W. Sperling, P. Lutz. Designing applications for an OSACA control // Proceedings of the International Mechanical Engineering Congress and Exposition, (The ASME Winter Annual Meeting). Dulles/USA, November 16-21, 1997. 5p.
2. www.osaca.org/documentation_and_software/demo_software.htm
3. www.omg.org/technology/documents/spec_catalog.htm
4. www.theaceorb.com/product/abouttao.html
5. D. C. Schmidt, M. Deshpande, C. O›Ryan. Operating System Performance in Support of Real-time Middleware // Proceedings of the 7th IEEE Workshop on Object-oriented Real-time Dependable Systems. San Diego, CA, January, 2002. P. 1-9.
6. www.cse.seas.wustl.edu/
7. www.theaceorb.com/downloads/index.html
8. www.embedix.com/
9. www.aero.polimi.it/~rtai/
10. www.fidia.it/english/research_ocean
11. R. Boissier, E. Gressier-Soudan, A. Laurent, L. Seinturier. Enhancing numerical controllers, using MMS concepts and a CORBA-based software bus. // International Journal of Computer Integrated Manufacturing, Publisher Taylor&Francis. Volume 14, № 6 (November). 2001, p.p. 560-569.

«СТРУЖКА» • e-mail: editors@struzhka.ru