Концепция числового программного управления мехатронными системами: технология компонентной организации программного обеспечения

МАРТИНОВ Г.М.
СОСОНКИН В.Л.
ФГУП НИИАЭ, Москва

Предложена компонентная организация программного обеспечения систем управления. Рассмотрены базовые понятия, представлены 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.

Введение. Высокие темпы эволюции технических требований к системам управления заставляют регулярно выпускать новые версии программного обеспечения; и 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 («run-time»); 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, 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.

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. 1).

Компонентная объектная модель COM, Component Object Model, делает стандартную структуру объекта регулярной, управляет его жизненным циклом и его общением с другими объектами [1-3]. Другими словами, COM определяет: правила структурирования объектов и их распределения в памяти, правила создания и уничтожения объектов, правила 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 [4]. Располагая указателем на IUnknown, клиент может запросить и другие интерфейсы компонента. Произ-

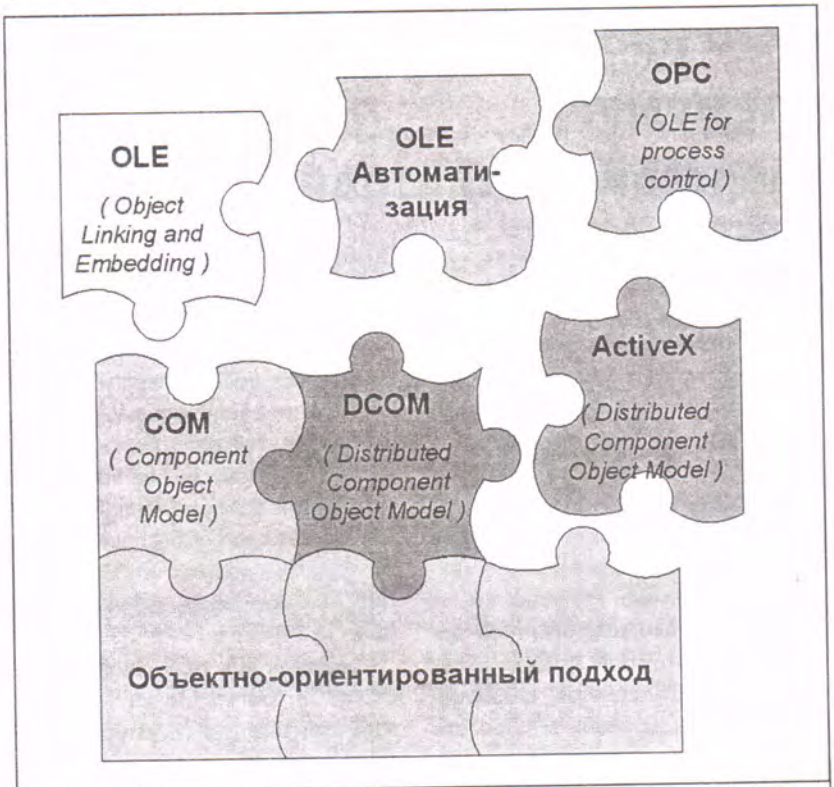

Рис. 1. Технологии разработки прикладного программного обеспечения систем управления.

водный от IUnknown интерфейс наследует три базовых метода. Метод QueryInterface() позволяет клиенту получить указатель на любой интерфейс компонента из другого указателя интерфейса. Методы Add() и Release() поддерживают 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 сервис 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. 2).

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 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.

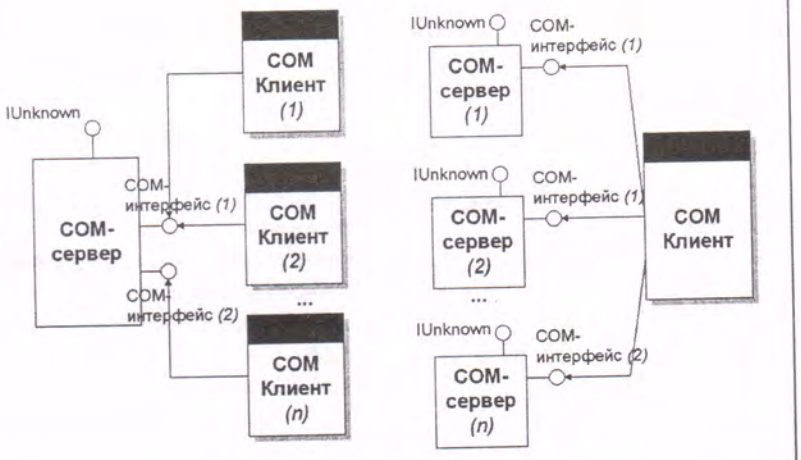

Рис. 2. Клиент-серверные отношения.

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, то есть 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 интерфейсом 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 не обязательно. Для ускорения используют стандартные библиотеки MFC (Microsoft Foundation Classes) или библиотеки шаблонного программирования ATL (Active Template Library) [5].

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 (то есть 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 внутренних 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();
```

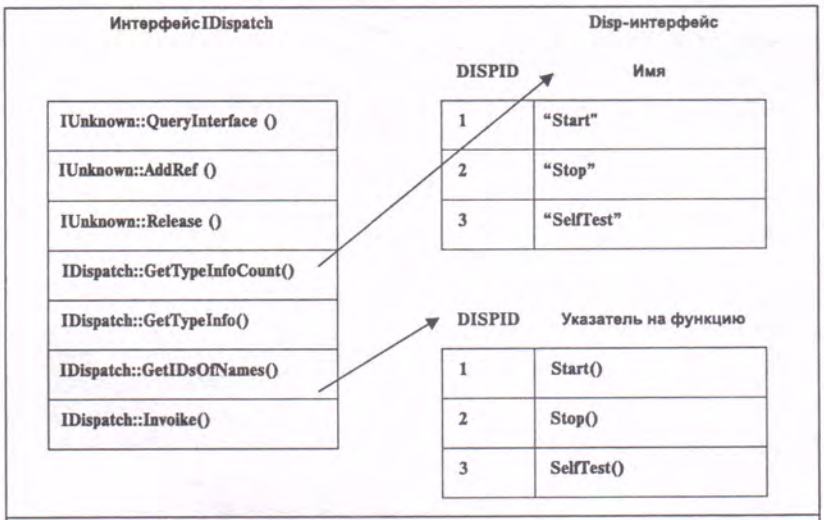

Рис. 3. Обобщенный интерфейс управления приводом на базе IDispatch.

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().

Классификация COM-интерфейсов и COM-серверов. При построении COM-сервера у разработчика есть альтернативы; в этой связи остановимся на классификации. Так, возможны следующие интерфейсы:

- 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() возвращает 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. 3). 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 рабо-

тают 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. 4.

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 устойчивость к ошибкам .

• Внепроцессный (out-of-process, local), предполагает, что клиент и сервер находятся на одном компьютере, но загружены в разные области его памяти (то есть 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 одного 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, по-

Таблица виртуальных функций объектов с интерфейсом IDriveControl	Таблица виртуальных функций объектов с дуальным интерфейсом IDriveControl
IUnknown::QueryInterface ()	IUnknown::QueryInterface ()
IUnknown::AddRef ()	IUnknown::AddRef ()
IUnknown::Release ()	IUnknown::Release ()
IDriveControl::Start()	IDispatch:: GetTypeInfoCount()
IDriveControl::Stop()	IDispatch::GetTypeInfo()
IDriveControl::SelfTest()	IDispatch::GetIDsOfNames()
	IDispatch::Invoike()
	IDriveControl::Start()
	IDriveControl::Stop()
	IDriveControl::SelfTest()

Рис. 4. Таблицы виртуальных функций объектов с интерфейсом IDriveControl.

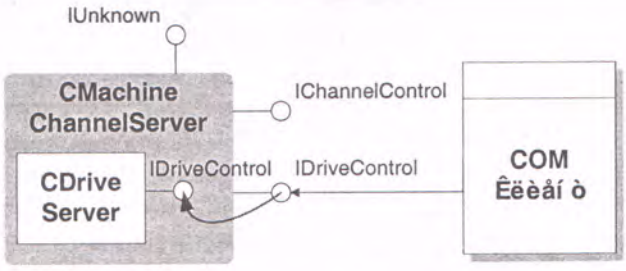

Рис. 5. Включение компонентов.

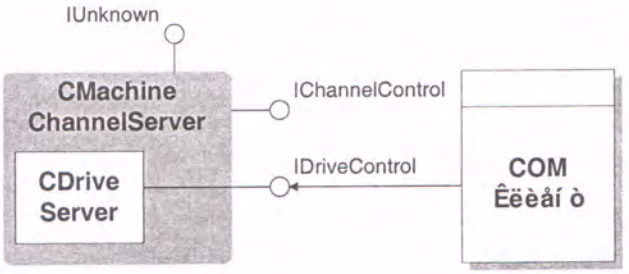

Рис. 6. Агрегация компонентов.

скольку 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. Возможны следующие способы использования одного компонента другим.

• Включение (containment) предполагает, что внешний компонент предоставляет интерфейс включаемому компоненту и обращается к нему для организации интерфейса. Пусть создаем компонент геометрического канала системы ЧПУ и вместо реализации функций управления приводом заново воспользуемся путем включения, компонентом управления приводом. Когда клиент обращается к интерфейсу IDriveControl (рис. 5), компонент геометрического канала CMachineChannelServer переправляет вызов компоненту CDriveServer. Внешний компонент может специализировать этот интерфейс, добавляя 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. 6. Она применяется тогда, когда реализация интерфейса устраивает разработчика полностью.

Область использования COM. Во-первых, это повторное использование компонентов. COM позволяет однажды создавать 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.

ния. 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-сер-

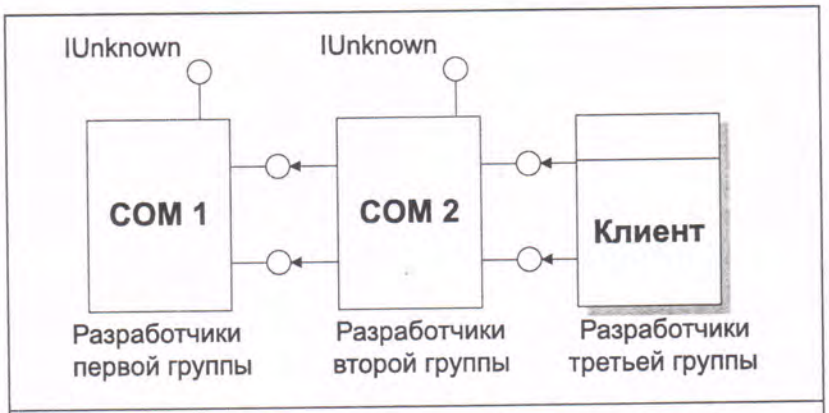

Рис. 7. Параллельная разработка программного обеспечения.

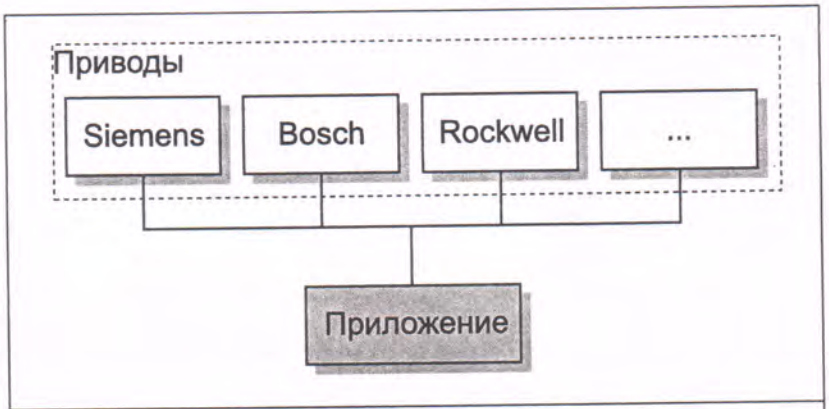

Рис. 8. Унификация программного обеспечения.

Во-вторых, это параллельная разработка. Обычно начинают с разработки интерфейсов компонента, что определяет корректность совместной работы всего программного обеспечения (рис. 7). Последующую разработку функциональных возможностей компонентов можно распараллеливать.

В-третьих, это унификация прикладного программного обеспечевер 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. 8).

Существуют некоторые особенности использования COM. Например, необходимо тщательно плани-

ровать 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.

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 [6].

Создадим абстрактный класс стандартного интерфейса IUnknown, который не будем генерировать (рис. 9). Объявим знакомые нам методы IUnknown как чистые виртуальные функции (pure function), то есть не имеющие реализации; 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 только заполнить функциями компонента, реализующими функциональность COM интерфейса.

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. 10. 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; взаимодействующий, с другой стороны, 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. ATL COM сервер, со своей стороны, через COM-интерфейс Igraphic_similator уведомляет графический эмулятор: об изменении текущего кадра управляющей программы, о возникновении и снятии ошибки в системе ЧПУ, об изменении конфигурации осей и каналов, о смене типа интерполяции в канале, о 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 осуществляла бы лишь синхронные вызовы, графический эмулятор вынужден был бы останавливаться и ждать ответа после каждого обращения к системе ЧПУ.

Выводы. Компонентная организация программного обеспечения систем управления позволяет повторно использование исходный код, применять готовые компоненты независимых поставщиков, имеющие-

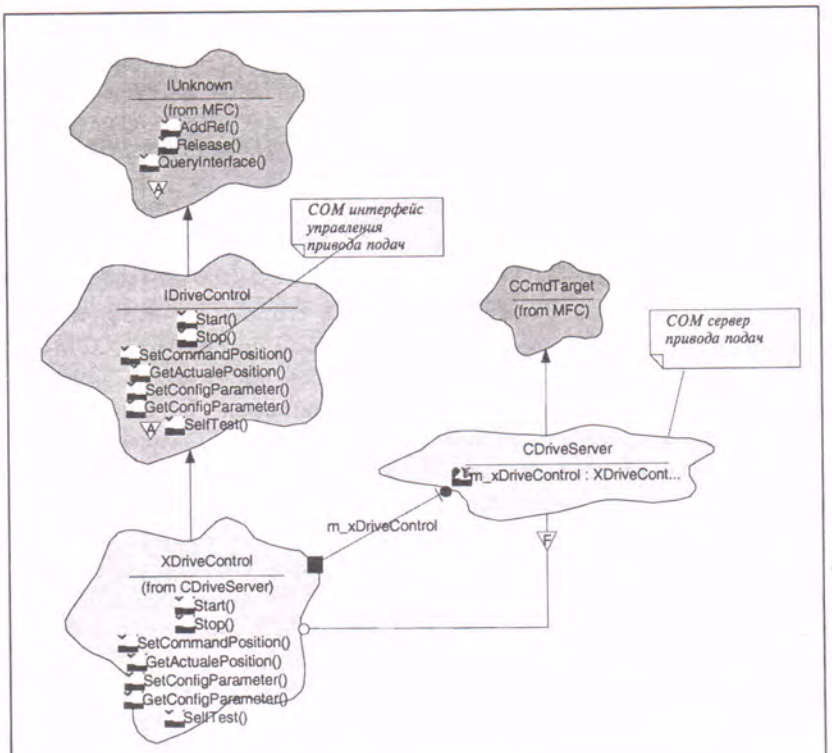

Рис. 9. Проектирование компонента в нотации Буча с помощью CASE-системы Rational Rose.

ся на рынке; компоновать систему управления под конкретные технологические задачи. Компонентный подход, применяемый на этапе проектирования программного обеспечения, позволяет 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 крупно-модульной реализации программного обеспечения систем числового программного управления мехатронными системами.

ЛИТЕРАТУРА:

1. Роджерсон Д. Основы COM / Пер. с англ. – М.: Издательский отдел «Русская Редакция» ТОО «Channel Trading Ltd.», 1997. – 376 с.

2. Оберг Р. Дж. Технология COM+. Основы и программирование. : Пер. с англ. Уч. пос. – М. : Издательский дом «Вильямс», 2000. – 480 с.

3. Рофэйл Эш, Шохауд Я. COM и COM+. Полное руководство: Пер. с англ. – К.: BEK +, К.: НТИ, М: Энтроп, 2000. 560 с.

4. Мартинов Г.М., Сосонкин В.Л. Концепция числового программного управления мехатронными системами: технология объектно-ориентированного программирования // Мехатроника, 2001, №7. С. 5-9.

5. Трельсен Э. Модель COM и применение ATL 3.0: Пер. с англ. – СПб.: BHV-Санкт-Петербург, 2000.-928 с.

6. Буч Г. Объектно-ориентированное проектирование с примерами приложений на C++, 2-е изд./Пер. с англ. / М.: «Издательство Бином», СПб: «Невский диалект», 1998 г. с. 560 с.

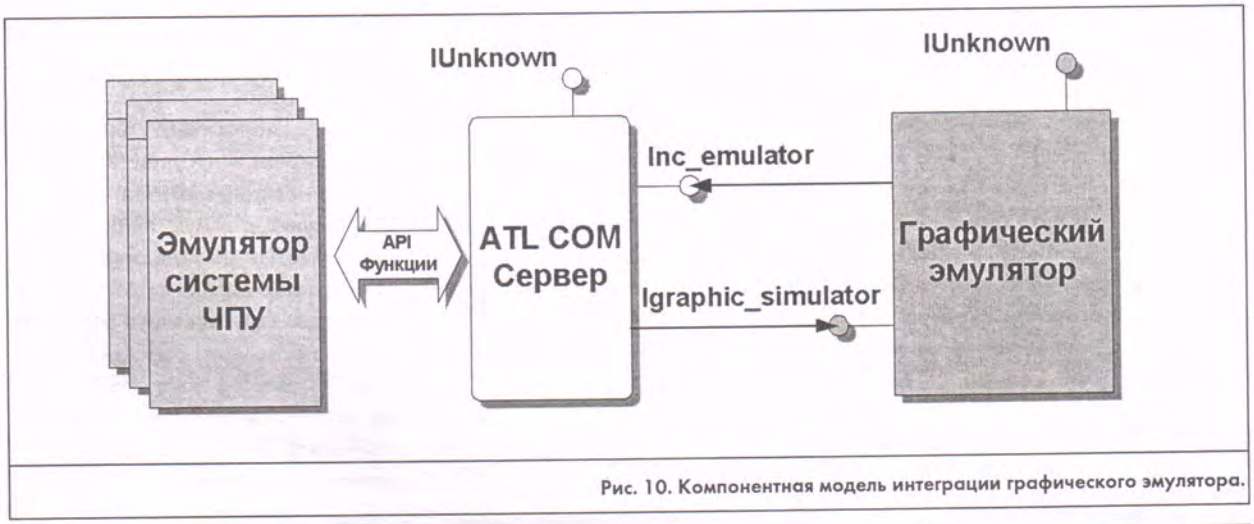

Рис. 10. Компонентная модель интеграции графического эмулятора.