Технология компонентной организации программного обеспечения

Базовые понятия

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.

- Компонентная модель COM лежит в основе таких технологий разработки прикладного программного обеспечения систем управления, как DCOM, OLE, OLE Автоматизация, ActiveX и OPC. Компонентная объектная модель COM делает стандартную структуру объекта регулярной, управляет его жизненным циклом и общением с другими объектами. 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

Технологии разработки прикладного программного обеспечения систем управления

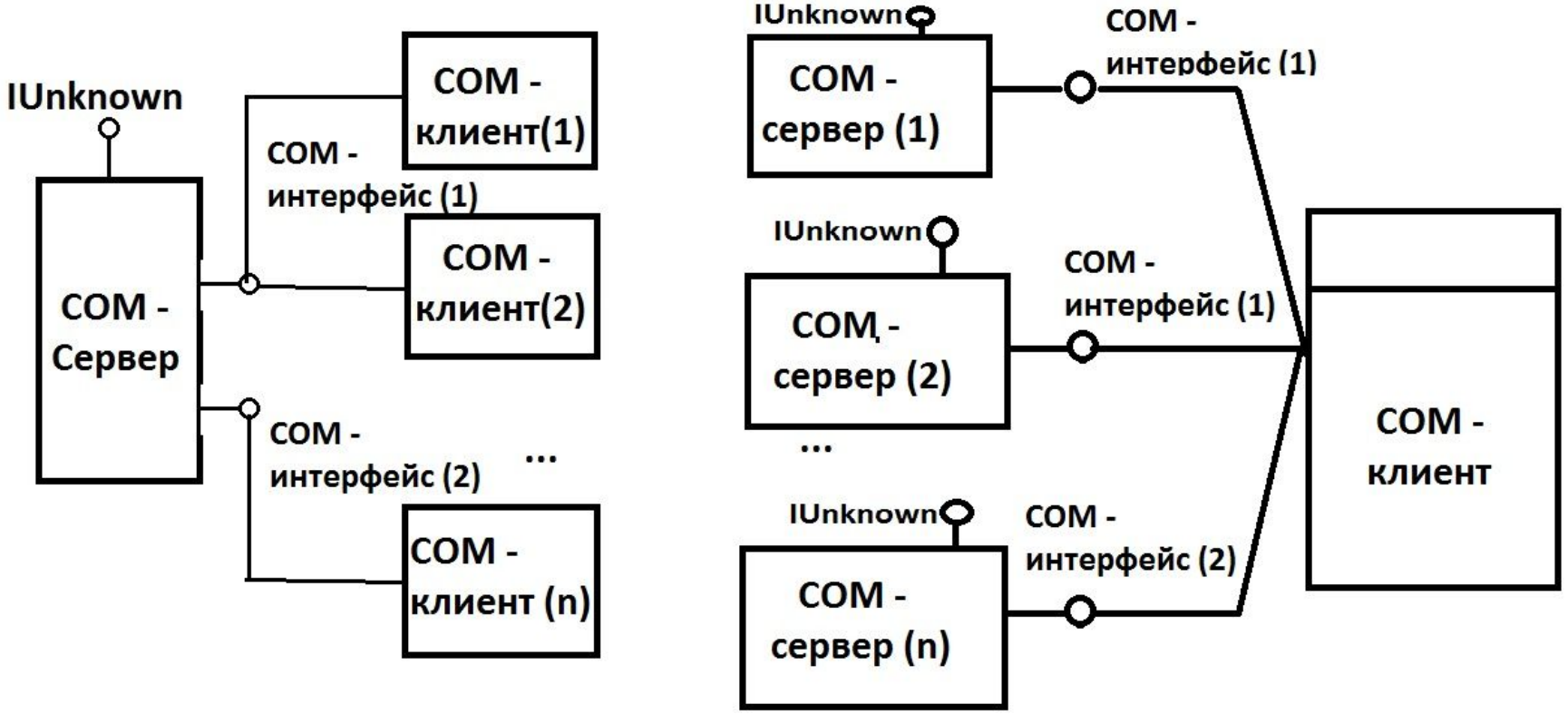
COM-интерфейс IUnknown

- 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. Располагая указателем на IUnknown, клиент может запросить и другие интерфейсы компонента. Производный от IUnknown интерфейс наследует три базовых метода:
- **QueryInterfaceO,** позволяющий клиенту получить указатель на любой интерфейс компонента из другого указателя интерфейса;
- **AddO** и **ReleaseO**, 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 *QueryInterfaceO* неявно вызывает метод *AddO* для увеличения содержимого счетчика, а после окон чания работы с интерфейсом клиент явно вызывает метод *ReleaseO* 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 .

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**.

Клиент-серверные отношения



- 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)**.

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 интерфейсом **IClassFactory2**, способный создавать компонент с идентификатором **CLSID**; она не рассчитана на реализацию интерфейсов создаваемого компонента.