Савва Юрий Болеславович

ПРОЕКТИРОВАНИЕ РАСПРЕДЕЛЕННЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

Лекция

Тема:

Модель COM/DCOM

Модель COM

- Основой как ActiveX, так и OLE является модель многокомпонентных объектов (COM).
- **COM устанавливает абстракции и правила, необходимые для определения объектов и интерфейсов; в ее состав входит также программное обеспечение, реализующее ключевые функции.** Хотя COM не является ни большой, ни слишком сложной, она определенно отличается как от других архитектур предоставления программных сервисов, так и от традиционных объектно-ориентированных подходов

Описание объектов COM

- **Программы, созданные с использованием COM, предоставляет свои сервисы через один или несколько COM-объектов. Каждый такой объект является экземпляром некоторого класса и поддерживает определенное количество интерфейсов, обычно не менее двух.**

- В состав каждого интерфейса входит один или более методов — функций, которые могут вызываться клиентом объекта.
- Например, один из гипотетических объектов, обсуждавшихся в предыдущей лекции, поддерживает как корректировщик орфографии, так и словарь синонимов, предоставляя доступ к своим сервисам через два разных интерфейса.
- Интерфейс корректировщика орфографии содержит LookUpWord, AddToDictionary и RemoveFromDictionary, тогда как интерфейс словаря синонимов состоит лишь из одного метода ReturnSynonym. Чтобы вызывать любой из этих методов, у клиента объекта должен быть указатель на интерфейс, содержащий соответствующий метод.

Интерфейсы

- Каждый поддерживаемый объектом интерфейс, по сути дела, — **контракт** между этим объектом и его клиентами. **Они обязуются: объект — поддерживать методы интерфейса в точном соответствии с определениями последнего, а клиент — корректно вызывать методы.** Чтобы контракт был работоспособным, объект и его клиенты должны договориться о способе явной идентификации каждого интерфейса, об общем способе спецификации — или описания — методов интерфейса, а также о конкретной реализации интерфейса.

Идентификация интерфейса

- У каждого интерфейса COM два имени.
- **Одно из них предназначено для людей — строка символов.**
- **Другое имя сложнее — оно предназначено для использования в основном программным обеспечением.**
- Легко воспринимаемое человеком символьное имя не является гарантированно уникальным — допускается (хотя это и не распространенная практика), чтобы это имя было одинаковым у двух интерфейсов.
- Имя же, используемое программами, уникально — это позволяет точно идентифицировать интерфейс.

- По соглашению читабельные **имена большинства COM-интерфейсов начинаются с буквы I** (от interface). Различные технологии, основанные на COM, определяют интерфейсы с разными именами, но все они обычно начинаются с буквы I и пытаются хотя бы немного описать назначение интерфейса. Например, интерфейс корректировщика орфографии, описанный выше, мог бы называться ISpellChecker, а интерфейс словаря синонимов — IThesaurus.

- Простые дружественные имена, вроде приведенных выше, **удобны при упоминании интерфейса** в разговоре или при выборе имен переменных и типов для указателей интерфейсов.
- Однако **они не годятся, когда клиент должен точно указать, какой именно интерфейс объекта ему нужен**. Например, если две группы независимо разрабатывают два разных интерфейса, может случиться, что для обоих интерфейсов будет выбрано имя ISpellChecker. Клиент, знающий только об одном из этих интерфейсов и запрашивающий у некоторого объекта указатель ISpellChecker, может получить неверный указатель, если объектом реализован в действительности другой интерфейс. А если объекту нужно реализовать оба интерфейса, его клиенты окажутся в затруднительном положении, так как не будут в точности знать, указатель какого интерфейса они получат, запрашивая ISpellChecker.

- **Создатель любого интерфейса должен присвоить ему уникальное имя — глобально уникальный идентификатор** (globally unique identifier — **GUID**).
- GUID интерфейса называется идентификатором интерфейса (interface identifier — IID).
- GUID — это 16-байтовая величина, обычно генерируемая программой-утилитой.
- Каждый может запустить такую утилиту на любом компьютере и гарантированно (со всех практических точек зрения) получит GUID, который будет отличаться от всех остальных.

Проблема сводится к тому, чтобы гарантировать уникальность каждого GUID во времени и пространстве. Уникальность во времени достигается за счет того, что каждый GUID содержит метку времени, указывающую, когда он был создан, что гарантирует отличие друг от друга всех GUID, сгенерированных на данной машине.

Для обеспечения уникальности в пространстве у каждого компьютера, который может быть использован для генерации GUID, должен быть уникальный идентификатор. В качестве такого идентификатора программа генерации GUID использует уникальное значение, уже имеющееся на большинстве компьютеров: адрес платы сетевого интерфейса.

Если в компьютере не установлен сетевой адаптер, то из различных случайных характеристик данной системы генерируется фиктивный идентификатор машины. Но и тогда маловероятно, что идентификаторы двух машин окажутся одинаковыми.

Хотя человеку и трудно работать с GUID, последние отлично подходят для назначения гарантированно уникальных имен интерфейсов — тех, что используются программами. Люди обычно выбирают для указания интерфейсов простые читабельные имена, а не GUID.

Не стоит забывать, однако, что у каждого интерфейса **фактически два имени — читабельное и IID** (который, конечно, по сути GUID) и что скомпилированное и работающее программное обеспечение практически всегда пользуется последним.

Спецификация интерфейса

- Объект и клиент должны иметь заранее согласованный способ описания интерфейса, т.е. способ определения методов, из которых состоит интерфейс, а также параметров этих методов.
- **COM не предписывает, как это должно быть сделано**. COM-объект может описывать свои интерфейсы с помощью чистого C++, или некоторого псевдо-C++, или каким-то еще способом, который может быть согласован между создателем объекта и создателями его клиентов.
- **Важно другое: COM-объект обязан точно следовать стандарту двоичного интерфейса COM.**

Для определения интерфейсов удобно иметь стандартный инструмент.

В COM такой инструмент есть — язык описания интерфейсов (Interface Definition Language — IDL).

IDL COM является в значительной степени расширением IDL Microsoft RPC, а тот в свою очередь заимствован из IDL OSF DCE (Open Software Foundation Distributed Computing Environ- j ment).

С помощью IDL можно составить полную и точную спецификацию интерфейсов объекта COM.

Например, ниже приведена спецификация на IDL для гипотетического интерфейса корректировщика орфографии ISpellChecker:

```
 [ object, uuid(E7CDODOO-1827-11CF-9946-444553540000) ]
interface ISpellChecker : Unknown
{ import "unknwn.idi"   HRESULT LookUpWord([in ] OLECHAR
word[31],[out] boolean * found);   HRESULT
AddToDictionary([in] OLECHAR word[31]):   HRESULT
RemoveFromOictionary([in] OLECHAR word[31]); }
```

Как можно заметить, IDL очень похож на C++.

- Спецификация интерфейса начинается со слова object, указывающего, что будут использоваться расширения, добавленные COM к оригинальному IDL DCE.
- Далее следует **IID интерфейса** — некоторый GUID. В DCE, откуда была заимствована эта идея, GUID называется универсально уникальным идентификатором (universal unique identifier — UUID).
- Так как в основе IDL COM лежит IDL DCE, то в описании интерфейса используется термин UUID.
- Иначе говоря: UUID — всего лишь другое имя для GUID.

Далее идет имя интерфейса — ISpellChecker, за ним — двоеточие и имя другого интерфейса — IUnknown.

Такая запись указывает, что ISpellChecker **наследует все методы, определенные для IUnknown,** т.е. клиент, у которого есть указатель на ISpellChecker, может также вызывать и методы IUnknown. IUnknown, как будет рассказано далее, является критически важным интерфейсом для COM, и все остальные интерфейсы наследуют от него.

COM поддерживает только наследование интерфейса, но не наследование реализации. Хотя объект COM может при определении своего интерфейса наследовать от любого из существующих интерфейсов, этот новый объект унаследует только само определение, но не реализацию существующего интерфейса.

На практике наследование интерфейсов используется в COM нечасто. Вместо этого объект обычно поддерживает каждый необходимый ему интерфейс по отдельности (кроме lUnknown, от которого наследуют все интерфейсы).

В отличие от C++ COM поддерживает лишь одиночное наследование, позволяя интерфейсу наследовать только от одного предка. Множественное наследование, т.е. наследование от нескольких интерфейсов одновременно, не поддерживается.

Программисты на C++ могут свободно применять множественное наследование C++ для реализации объектов COM, однако оно не может быть использовано для спецификации интерфейсов на IDL.

Далее в спецификации интерфейса идет оператор import. Так как данный интерфейс наследует от IUnknown, то некоторой программе, читающей определение интерфейса, может потребоваться найти описание IDL для IUnknown. Оператор import указывает такой программе, какой файл содержит нужное описание.

Вслед за оператором import в описании интерфейса идут три метода: LookUpWord, AddToDictionary и RemoveFromDictio-nary, — а также их параметры.

Все три возвращают **HRESULT — стандартное возвращаемое значение**, указывающее, был ли вызов обработан успешно.

Параметры в IDL могут быть сколь угодно сложными, использовать такие типы, как структуры и массивы, являющиеся производными своих аналогов в C++.

Каждый параметр помечен [in] или [out]. **Значения параметров [in] передаются при вызове метода от клиента объекту, тогда как значения [out] передаются в обратном направлении.** (Параметры, значения которых передаются в обоих направлениях, могут быть помечены как [in, out].) Подобные метки могут помочь читателю лучше понять интерфейс, однако их основное назначение в том, чтобы программа, обрабатывающая спецификацию интерфейса, точно определила, какие данные и в каком направлении копировать.

Подобная простая спецификация — все, что необходимо для заключения контракта между объектом COM и его клиентом. Хотя интерфейс и не обязан задаваться именно таким способом, но следование ему может значительно облегчить жизнь разработчика. Кроме того, неплохо использовать одну стандартную схему для спецификации интерфейсов объектов COM.

- Важное обстоятельство, связанное со спецификациями интерфейса: **после того как интерфейс опубликован и начал где-то работать, после того как его задействовали в выпущенной версии какого-либо программного обеспечения, изменять его, по правилам COM, нельзя**.
- Если создатель интерфейса хочет добавить новый метод, изменить список параметров метода или внести какие-либо другие изменения, это также недопустимо.
- **Интерфейсы должны быть фиксированными.**

- **Добавление новой или изменение существующей функциональности требует определения полностью нового интерфейса.**
- Такой интерфейс может наследовать от старого, но тем не менее является совершенно отличным от него и имеет новый и другой IID.
- Создатель программного обеспечения, поддерживающего новый интерфейс, может продолжать поддерживать и старый, но это не является требованием.
- Создатель программы имеет право прекратить поддержку старого интерфейса (хотя обычно это плохая идея), но ему **категорически запрещено изменять этот интерфейс**.

Реализация интерфейса

- Чтобы вызвать метод, клиенту необходимо точно и подробно знать, как это делать. Спецификация интерфейса, подобная приведенной выше, описывает лишь одну важную часть процесса.
- **Но COM также определяет и другое: она задает стандартный двоичный формат, который каждый COM-объект должен поддерживать для каждого интерфейса. Наличие стандартного двоичного формата означает, что любой клиент может вызывать методы любого объекта независимо от языков программирования, на которых написаны клиент и объект.**

- **Клиентский указатель интерфейса фактически является указателем на указатель внутри объекта. Последний в свою очередь указывает на таблицу, содержащую другие указатели.**
- **Эта виртуальная таблица (viable) содержит указатели на все методы интерфейса.**

- Что представляют собой первые три метода ISpellChecker? Это методы, определенные интерфейсом IUnknown. Поскольку ISpellChecker наследует от IUnknown, у клиента должна быть возможность вызова методов lUnknown через указатель на ISpellChecker. Чтобы это стало возможным, виртуальная таблица ISpellChecker должна содержать указатели на эти три метода.
- **Фактически, так как каждый интерфейс наследует от IUnknown, виртуальная таблица любого COM-интерфейса начинается с указателей на три метода IUnknown. Подобная двоичная структура имеет место для всех интерфейсов, поддерживаемых любым объектом.**

- **Формат интерфейса COM моделирует структуру данных, генерируемую компилятором C++ для класса этого языка** (класс задает тип объектов, а двоичная структура генерируется для объектов класса.).
- Это сходство означает, что COM-объекты очень легко создавать на C++. Хотя COM-объекты можно писать на любом языке, поддерживающем описанные стандартные двоичные структуры, в COM имеется некоторый уклон в сторону реализации на C++.

- **При вызове клиентом метода интерфейса выполняется проход по описанной структуре (с помощью указателя на виртуальную таблицу извлекается указатель на метод, который в свою очередь извлекает код, фактически предоставляющий сервис) и исполняется соответствующий код.**
- Если клиент написан на C++, этот проход невидим для программиста, поскольку C++ и так делает это автоматически. Вызов методов COM из программы на C несколько сложнее. Тот, кто пишет клиент на C, должен знать, что необходимо пройти по цепочке указателей, и кодировать вызов соответствующим образом. Результат в любом случае один и тот же: исполняется метод в объекте.

IUnknown — фундаментальный интерфейс

- **Каждый объект COM должен поддерживать интерфейс IUnknown — в противном случае он не будет объектом COM.**
- IUnknown содержит только три метода: QueryInterface, AddRef и Release.
- Так как все интерфейсы наследуют от IUnknown, его методы могут быть вызваны через любой из указателей на интерфейсы объекта. Тем не менее IUnknown является отдельным самостоятельным интерфейсом с собственным IID, так что клиент может запросить указатель непосредственно на IUnknown.
- На диаграммах lUnknown обычно изображается над объектом.

Назначение IUnknown::QueryInterface

- Обычно свой первый указатель на интерфейс объекта клиент получает при создании объекта.
- Имея первый указатель, клиент может получить указатели на другие интерфейсы объекта, методы которых ему необходимо вызывать. Для этого **клиент просто запрашивает у объекта эти указатели с помощью IUnknown::QueryInterface**.

Чтобы воспользоваться QueryInterface, клиент вызывает его с помощью любого из имеющихся у него в данный момент указателей на интерфейсы объекта. Клиент передает IID нужного ему интерфейса как параметр метода.

Например, пусть у клиента уже имеется указатель на интерфейс A, и требуется получить указатель на интерфейс B. Клиент запрашивает данный указатель вызовом QueryInterface через указатель A, задавая в качестве параметра IID интерфейса B (шаг 1). Если объект поддерживает B, то он возвращает указатель на этот интерфейс (шаг 2), и клиент может теперь может вызывать методы B (шаг 3). Если же объект не поддерживает B, он возвращает NULL.

Самый важный элемент COM — QueryInterface.

Именно эта простая **схема решает очень важную и сложную проблему — контроль версий**.

Объекты, составляющие такие приложения, создаются множеством организаций, каждая из которых модернизирует свои объекты независимо от остальных. Как все это будет работать, если новые возможности добавляются в разные объекты в разное время? Как установить новую версию объекта с расширенными возможностями, не повредив программам, использующим только старые возможности? И как после модернизации клиента под новые возможности обеспечить автоматическое начало их использования этим клиентом? Ответ на все эти вопросы дает QueryInterface.

Лучше всего продемонстрировать это на примере.

Допустим, имеется некий набор инструментов обработки текста, реализованный в виде COM-объекта, поддерживающего интерфейс ISpellChecker.

Если установить такой объект на компьютер, текстовый процессор (и другие клиенты) сможет его использовать.

Чтобы получить доступ к сервисам объекта, текстовый процессор запрашивает указатель на ISpellChecker через QueryInterface.

Так как объект поддерживает этот интерфейс, то возвращает соответствующий указатель, и текстовый процессор вызывает методы ISpellChecker.

Теперь допустим, что фирма, продающая этот объект — инструментарий для обработки текста, — решила добавить поддержку словаря синонимов, доступ к которой можно получить через интерфейс IThesaurus.

Таким образом, следующая версия объекта поддерживает как ISpellChecker, так и IThesaurus. После того, как установить на машине эту новую версию, все будет работать так же, как и раньше. Текстовый процессор, как обычно, запрашивает указатель на ISpellChecker и успешно пользуется его методами (вспомним, COM запрещает изменение интерфейсов.) То, что объект теперь поддерживает еще и IThesaurus, совершенно неизвестно "ограниченному" текстовому процессору, так как он не поддерживает работы со словарем синонимов.
Следовательно, старый текстовый процессор никогда не запросит у объекта указатель на этот интерфейс.

Предположим теперь, что на машине установлена новая версия текстового процессора, поддерживающая работу со словарем синонимов. Когда в следующий раз пользователь вызовет старый текстовый процессор, он, как обычно, запустит объект — инструментарий для обработки текста и запросит указатель на интерфейс ISpellChecker. Однако новая версия текстового процессора обладает информацией, достаточной для того, чтобы запросить указатель на IThesaurus. Так как версия объекта, которая поддерживает данный интерфейс, была установлена ранее, нужный указатель будет возвращен, и текстовый процессор сможет воспользоваться новой функцией.

Итак, в итоге установлена новая версия инструментария для обработки текста, не нарушающая при этом работы существующих его клиентов, а также обеспечено автоматическое использование этими клиентами функций новой версии, после того как сами клиенты были обновлены.

Ну а как быть тем, кто установил новую версию текстового процессора, но еще не приобрел новую версию инструментария для обработки текста?

Все также работает за исключением того, что текстовый процессор не предоставляет таким пользователям возможностей словаря синонимов. Текстовый процессор запускает объект-инструментарий и через QueryInterface успешно получает указатель на ISpellChecker. Однако, запрашивая указатель на IThesaurus, он получает в ответ NULL. Если текстовый процессор написан с учетом подобной возможности, он отключает пункт меню Thesaurus. Поскольку объект, реализующий IThesaurus, отсутствует, постольку у пользователя не будет доступа к функциям словаря синонимов. Как только пользователь потратится на модернизированный объект — инструментарий для обработки текста, этот пункт меню будет активизирован без каких-либо изменений в текстовом процессоре.

Рассмотрим еще один пример. Что, если создатель объекта — инструментария для обработки текста — пожелает изменить или расширить функциональные возможности объекта по корректировке орфографии? Это влечет за собой изменение или добавление новых методов, которые будут видимы клиенту объекта.

Однако COM не разрешает изменять интерфейсы, поэтому существующий интерфейс ISpellChecker трогать нельзя. Вместо этого создатель объекта должен определить новый интерфейс, скажем, ISpellChecker2, и включить в него необходимые новые или измененные методы.

Объект по-прежнему поддерживает ISpellChecker, но теперь он также будет поддерживать и ISpellChecker2. Добавление в объект поддержки ISpellChecker2 ничем не отличается от добавления поддержки любого нового интерфейса. Как и все COM-интерфейсы, новый имеет уникальный IID, который клиент, знающий о новом интерфейсе, может использовать для запроса указателя через QueryInterface. Как и в предыдущем случае с IThesaurus, клиенты, ничего не знающие о происшедшей модернизации, никогда не запросят указатель на ISpellChecker2, и не ощутят никакого воздействия со стороны изменений — они будут продолжать использовать ISpellChecker, как прежде.

QueryInterface и требование неизменности интерфейсов COM позволяют программным компонентам, разрабатываемым независимыми организациями, обновляться по отдельности и тем не менее продолжать нормальную совместную работу. Это соображение трудно считать несущественным, и именно поэтому создатели COM иногда называют QueryInterface важнейшим элементом модели.

Подсчет ссылок