دانشگاه صنعتی اصفهان

دانشکده برق و کامپیوتر

گزارش پروژه درس شبکه های کامپیوتری

مبانی DCOM

علی بهلولی زفره

محمد اسماعیل جعفری

سید علی امام

استاد راهنما :

دکتر فرامرز هندسی

1380

فهرست

1- مبانی DCOM :

قبل از هر چیز باید با لغات به کاربرده شده در مورد **DCOM(Vocabulary)** و مفاهیم مربوط به آن صحبت کنیم. برای درک این مطالب ، مفاهیم و لغات مربوط به اشیاء **C OM** را با اشیاء معمولی **C++** مقایسه می کنیم و تفاوتها و شباهتهای آنها را توضیح خواهیم داد.

1-1- کلاسها و اشیاء :

فرض کنید که یک کلاس ساده از **C++** به نام **xxx** ایجاد کرده ایم که چندین تابع عضو (**member function**) به نامهای **method A** ، **method B** و **method C** دارد. هر کدام از توابع عضو پارامترهایی می گیرند و مقداری را برمی گردانند. تعریف کلاس **(declearation)** را در زیر می‌بینید:

```
class xxx {
public:
int     methodA(int     a);
int     methodB(float   b);
float   methodC(float   c);
};
```

تعریف کلاس به خودی خود کلاس را تشریح می کند. هنگامیکه بخواهید از کلاس استفاده کنید، باید یک **instance** از این شیء ایجاد کنید. **Instance** ها اشیاء واقعی هستند، اما کلاسها تنها تعریف (**definition**) هستند. هر شیء یا به عنوان یک متغیر(محلی یا **global**)ایجاد می شود یا به صورت دینامیک با استفاده از کلمه کلیدی **new** متغیری را به صورت دینامیک در فضای حافظه **heap** ایجاد می کند و یک اشاره گر به آن برمی گرداند. هنگامیکه توابع عضو را صدا می‌زنید، با استفاده از اشاره گر این کار را انجام می‌دهید. به عنوان مثال :

```
xxx *px;                    // pointer to xxx class
px = new xxx;               // create object on heap
px -> methodA(1);           // call method
delete px;                  // free object
```

باید بدانید که **com** نیز از مدل شیء گرای مشابهی پیروی می‌کند. **Com** مانند اشیاء **C++** دارای **class**، توابع عضو و تعریف **instance** است. با اینکه شما هیچگاه **new** را برای ایجاد اشیاء **com** صدا نی‌زنید، باید آنرا در حافظه ایجاد کنید. همچنین به اشیاء **com** با اشاره گر دسترسی پیدا می‌کنید و هنگامیکه کارتان به پایان رسید باید حافظه را پس بگیرید **(dealocate)**

هنگامیکه که کد **com** می‌نویسیم از **new** و **delete** استفاده نمی‌کنیم. با وجود اینکه از **C++** به عنوان زبان برنامه نویسی استفاده می کنیم، یک **syntax** کاملاً جدید خواهیم داشت. **Com** با فراخوانی یک **com API** ایجاد می‌شود (**com API** امکان ایجاد و تخریب اشیاء **com** را فارهم می‌کنند). به **pseude-code** زیر توجه کنید :

```
ixx    * pi;                          // pointer    to xxx com interface
        CoCreate Instance( , , , , &pi );      // create interface
            Pi -> methodA(1);                  // call method
          Pi -> felease( );                    // free interface
```

در این مثال ما کلاس **I x x** را **"interface"** نامیدیم. متغیر **pi** اشاره گری به **interface** است. تابع **CoCreateIntstance** یک **instance** از جنس **ixx** ایجاد می کند. این اشاره‌گر **interface** برای فراخوانی توابع استفاده می‌شود. **Release**، **interface** را از بین می برد.

پارامترهای تابع **CoCreateIntstance** در مثل بالا، عملاً حذف شده اند تا مثال ساده باشد. این پارامترها به توضیح بیشتری نیتز دارند، که بعداً به آن خواهیم پرداخت.

1-2- تفاوتهای اشیاء COM با اشیاء C++ :

اشیاء **COM** پیچیده‌تر از اشیاء **C++** هستند. برخی از این پیچیدگیها بخاطر ملاحضات شبکه‌ای هستند. 4 فاکتور اصلی زیر ما را به طراحی اشیاء **C OM** وادار می‌کند:

اشیاء **C++** همیشه در یک فضای فرایند (**process space**) اجرا می‌شوند در حالی که اشیاء **COM** می‌توانند در فرایندهای مختلف یا در کامپیوترهای مختلف اجرا شوند.

متدهای **COM** می‌توانند روی شبکه فراخوانی شوند.

نام متدهای اشیاء **C++** در فضای فرایند فعلی باید واحد باشند. نام اشیاء **COM** باید در جهان واحد باشد.

اشیاء سرور **COM** می‌توانند به زبانهای گوناگون و روی سیستم‌عاملهای کاملاً متفاوت نوشته شوند.

اجازه دهید ببینیم این تفاوتهای بین اشیاء **COM** و اشیاء **C++** برای شما به عنوان برنامه‌نویس چه معنایی می‌تواند داشته باشد.

1-3- اشیاء COM می توانند در فرایندهای جداگانه‌ای اجرا شوند:

به کمک **COM** شما می‌توانید در فرایند دیگری یا روی ماشین دیگری در شبکه اشیاء مورد نظر خود را ایجاد کنید. این امکان، به این معناست که شما نمی‌توانید اشیاء را به کمک دستور **new** در **C++** ایجاد کنید و برای فراخوانی متدهای آن، فراخوانی محلی کافی نیست. برای ایجاد اشیاء **COM**، یک موجودیت اجرایی(فایل **Exe** یا سرویس) باید فضاگیری از راه دور و ایجاد شیء را بر عهده بگیرد. این وظیفه پیچیده‌ای است. این مشکل با ایجاد مفهوم **server COM** حل می‌شود. این شیء جدید باید با **client** ارتباط ایجاد کند.

1-4- متدهای COM می‌توانند روی شبکه فراخوانی شوند :

اگر روی شبکه به ماشینی دسترسی داشته باشید و**COM server** شیئی که می‌خواهید از آن استفاده کنید روی آن کامپیوتر نصب شده است، شما می‌توانید یک شیء **COM** روی آن ماشین ایجاد کنید. مسلماً شما باید اجازه‌های معمول را داشته باشید و شیء **COM server** روی کامپیوتر مقابل به درستی نصب شده باشد. از آنجا که شیء **COM** مورد نظر شما لزوماً نباید روی ماشین محلی‌تان باشد، راه خوبی برای اشاره کردن به آن مورد نیاز است هر چند حافظه مورد نظر در جای دیگری قرار دارد. از نظر تکنیکی هیچ راهی برای این کار وجود ندارد و باید با مفهوم جدیدی شبیه سازی شود. مفهوم جدیدی که **COM** از آن استفاده می‌کند، **proxy/stub** است که ما بعداً آنرا توضیح خواهیم داد.

پیامد دیگر، ارسال داده بین **COM client** و**COM server** مربوطه‌اش می‌باشد. ارسال داده بین فرایندها یا نخها روی شبکه، **"Marshaling"** نامیده می‌شود. بار دیگر **proxy/stub** وظیفه **Marshaling** را برای شما انجام می‌دهد. **Com** همچنین می‌تواند برای **marshaling** در **interface**های خاصی از **type libraries** و **Automatic Marshaller** کند. لازم نیست **Automatic Marshaller** برای هر **COM server** به طور سفارشی ساخته شود.

1-5- اشیاء COM باید در جهان یکتا باشند :

این موضوع در نگاه اول غلوّ به نظر می‌رسد. اما در نظر داشته باشید که **I nternet** یک شبکه جهانی است. حتی اگر شما روی یک کامپیوتر منفرد کار می‌کنید، **com** باید احتمالات زیر را در نظر بگیرد.

یکتا بود الزامی است. در **C++**،به کمک کامپایلر تمام کلاسها به طور یکتا مورد استفاده قرار می‌گیرند. کامپایلر می‌تواند تعریف کلاس را برای تمامی کلاسهای استفاده شده در برنامه ببیند و همه اشاره‌گرها را با آنها تطبیق دهد و از تطابق آنها اطمینان حاصل کند. کامپایلر همچنین می‌تواند تضمین کند که یک کلاس با یک نام خاص وجود دارد. در **COM** باید راه مناسبی برای رسیدن به یکتایی مشابهی وجود داشته باشد. **COM** باید تضمین کند که تنها یک شیء با یک نام مشخص وجود دارد با اینکه اشیاء موجود در شبکه جهانی بسیار زیادند. این مشکل با ایجاد مفهومی به نام **GUID** حل می‌شود.

1-6- COM مستقل از زبان پیاده‌سازی است :

COM server ها ممکن است با زبانهای مختلف و روی سیستم عاملهای مختلفی نوشته شوند. اشیاء **COM** توانایی دسترسی از راه دور را دارند. این بدان معناست که آنها ممکن است روی **thread** ها، پروسسها یا حتی کامپیوترهای مختلفی باشند. کامپیوتر دیگر ممکن است حتی تحت سیستم عامل دیگری کار کند. پس نیاز به راه مناسبی برای ارسال پارامترها روی شبکه به شیء دیگری روی کامپیوتر دیگری وجود دارد. این مشکل با ایجاد راه جدیدی برای تعیین دقیق **interface** بین **client** و **server** حل می‌شود. همچنین کامپایلر جدیدی به نام **MIDL (Microsoft Interface Definition Language)** لازم است.این کامپایلر تعریف عمومی (**generic**) **interface** بین **client** و **server** را ممکن می‌سازد. **MIDl** ، اشیا، **COM** ، **interface** ها، متدها و پارامترها را تعریف می‌کند.

1-7- فرهنگ لغات COM :

یکی از مشکلاتی که ما با آن درگیر هستیم آن است که لغات کلیدی در معانی متفاوتی به‌کار می‌روند. شما احتمالاً با لغات مربوط به **Object Oriented** و **C++** آشنا هستید. جدول زیر مفاهیم لغات معادل قدیمی و **COM** را به نمایش می‌گذارد.

Concept	Convertional(C++/oop)	COM
Client	برنامه‌ای که یک متد**COM** را فراخوانی می‌کند	برنامه‌ای که یک سرویس را از سرور تقاضا می‌کند
Server	برنامه‌ای که برنامه‌های دیگری را سرویس می‌دهد	برنامه‌ای که شیء **COM** را برای **COM client** آماده می‌کند
Interface	وجود ندارد	اشاره‌گر به گروهی از توابعی که از طریق **COM** فراخوانی می‌شوند
Class	یک **data type** که که گروهی از قیدها و داده‌ها را تعریف می‌کند که با هم استفاده می‌شود	تعریف یک شیء که یک یا چند **COM interface** را پیاده‌سازی می‌کند. همچنین **coclass**
Object	یک **instance** از یک کلاس	یک **interface** از **coclass**
Marshalling	وجود ندارد	انتقال **data** بین **client** و **server**

توجه دارید که مفاهیم دارید **interface** و **Marshalling** به مدل **C++** قابل ترجمه نیستند. نزدیکترین چیز به **interface** در **C++** تعریف خارجی یک **dll** است. **Dll** بسیاری از کارهای مشابه یک شیء **COM** را، وقتی که **COM server** و **COM client** بسیار نزدیک به هم(در یک پروسس) هستند، انجام می‌دهد. **Marshalling** در **C++** اغلب کاملاً دستی صورت می‌گیرد. اگر بخواهید اطلاعات را بین پروسسها و کامپیوتر کپی کنید، باید کدی بنویسید که از برخی انواع ارتباط بین فرایندی استفاده می‌کند. برای این کار شما انتخابهای بسیاری خواهید داشت شامل **socket**ها، **clipboard** و **mailslot** ها.

1-8- **Interface** :

تاکنون لغت **"interface"** را به ندرت به کار برده‌ایم. در فرهنگ لغت **interface** به این صورت معنا شده است : « سطحی که به عنوان مرز مشترک بین دو جسم یا دو سطح تلقی می‌شود». این واقعاً یک توصیف مفید است. در **COM** ، **interface** یک معنای خیلی مخصوص دارد. **COM interface** یک مفهوم کاملاً جدید است که در **C++** وجود نداشت. درک مفهوم **interface** برای بسیاری از مردم ، در برخورد اولیه ، مشکل است. **Interface** یک موجود میهمان است که هرگز وجود مستقل ندارد . به عبارت دیگر شبیه یک کلاس مجرد (**abstract class**) است ، اما دقیقاً آن

نیست. به ساده‌ترین صورت ، **interface** چیزی جز مجموعه‌ای نامدار از توابع نیست . در **C++** کلاس تنها می‌تواند یک **interface** داشته باشد. توابع عضو این **interface** تمام توابع عضو **public** کلاس هستند. به‌عبارت دیگر ، **interface** قسمتی از کلاس است که برای همه قابل رویت است. همچنین در **C++** تفاوتی بین **interface** و کلاس وجود ندارد. در اینجا یک مثال از کلاس **C++** را می‌بینید :

```
class  yyy {
        public :
int     DOThis( );
        private :
void    Help( );
int     count;
int     x,y,z;
        } ;
```

هنگامیکه کسی می‌خواهد از این کلاس استفاده کند ، تنها به اعضای **p ublic** آن دسترسی دارد. (فعلاً اعضای **p rotected** و ارث بری را در نظر نمی‌گیریم.) این اشخاص نمی‌توانند **Help** را فراخوانی کنند و از هیچ کدام از متغیرهای **private** نمی‌توانند استفاده کنند. برای استفاده کننده از این کلاس، تعریف کلاس مشابه تعریف زیر به نظر می‌رسد :

```
class       yyy {
int     DoThis( );
        } ;
```

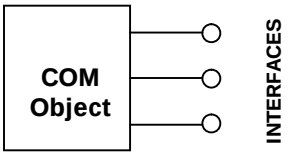
این زیر مجموعه **public** از کلاس، **interface** آن به دنیای خارج است. اصولاً **interface** ، داخل کلاس را از استفاده کننده مخفی می‌کند.

این مقایسه با **C++** بسیار دور از واقعیت است. **COM interface** یک کلاس **C++** نیست. **COM interface** برای خود قوانین و رسومات جداگانه‌ای از کلاسهای **C++** دارد.

مدل **COM** اجازه می‌دهد که **coclass** (**COM class**)، چند **interface** داشته‌باشد که هر **interface** نام و مجموعه توابع خاص خود را دارد. دلیل این خاصیت این است که بتوانیم اشیاء پیچیده تر و کاراتری داشته باشیم. این مفهوم دیگری است که معادلی ذز **C++** ندارد.(شاید چندین **interface** را بتوان به عنوان **union** دو تعریف کلاس در نظر گرفت که در **C++** امکان ندارد.)

1-9- Interface ، client را از server ایزوله می کند :

یکی از قوانین اصلی استاندارد **COM** این است که شما فقط از طریق **interface** می‌توانید به شیء **COM** دسترسی داشته باشید. برنامه **client** به وسیله **interface** کاملاً از **server** ایزوله شده است. این یک نکته بسیار مهم است. برنامه **client** هیچ چیزی راجع به شیء **COM** یا کلاس **C++** که شیء **COM** براساس آن پیاده‌سازی شده است، نمی داند و تمام چیزی که می‌تواند ببیند **interface** است. **Interface** شبیه پنجره‌ای به شیء **COM** است. طراح **interface** به **client** فقط اجازه دیدن قسمتهایی از شیء را که او می‌خواسته آشکار باشند، می‌دهد. شکل روبرو نشان می‌دهد که چگونه همه دسترسی‌های **client** ها به شیء **COM** از طریق **interface** انجام می‌شود.



علائمی که در اینجا استفاده شده است (دایره کوچکی که به وسیله خط به شیء وصل است)، روش مرسوم نمایش **interface** اشیاء **COM** است. نکات مهم زیادی در مورد **interface** ها وجود دارد ولی فعلاً برای پرداختن به موضوع مهم‌تر چگونگی کار اشیاء **COM** به مفهوم کلی **interface** قناعت می کنیم.

1-10- تصور کردن interface یک شیء COM :

راه دیگری برای تصور کردن **interface** وجود دارد. در این بخش ما **COM interface** را بدون هیچ‌کدام از مباحث مربوط به **C++** مطرح می کنیم. تلاش می کنیم که به **interface** به شکل مجرد (**abstract**) آن نگاه کنیم. یک اتومبیل را تصور کنید. همه اشیاء "اتومبیل" که در دنیای واقعی می‌شناسید یک **interface** "راندن" دارند که اجازه می‌دهند اتومبیل را به راست یا چپ هدایت کنید و نیز سرعت آنرا زیاد یا کم نمایید. توابع عضو برای **interface** راندن می‌توانند **left** ، **right** ، **faster** ، **slower** ، **forward** ، **reverse** باشند. بسیاری از ماشینها همچنین **interface** دیگری برای **radio** دارند. توابع عضو **interface** رادیو می‌توانند **on** ، **off** ، **louder** ، **softer** ، **next station** و **previous station** باشند.

driving	radio
Left() Right() Slower() Faster() Forward() Reverse()	On() Off() Loader() Softer() Next station() Prv. station()

اتومبیلهای زیادی وجود دارد، اما همه آنها رادیو ندارند. و به همین جهت **interface** را دیو را ایجاد نمی کنند. اما **interface** راندن را حمایت

می‌کنند. در همه ماشینهایی که رادیو وجود دارد تواناییهای رادیو یکسان است.

COM گونه مشابهی از همین مدل را برای کلاسهای **COM** حمایت می‌کند. اشیاء **COM** می‌توانند مجموعه‌ای از **interface** ها را حمایت کنند که هر کدام نام مشخصی دارند. برای اشیاء **COM** که شما خودتان ایجاد می‌کنید، شما اغلب یک **interface** را تعریف و استفاده می‌کنید. اما تعداد زیادی از اشیاء **COM** موجود چندین **interface** براساس خواصی که حمایت میکنند دارند.

یکی دیگر از مسایل مهم این است که **interface** راندن ، خود اتومبیل نیست. **Interface** راندن هیچ چیز درباره ترمزها ، چرخها یا موتور اتومبیل به شما نمی‌گویند. به عنوان مثال شما موتور اتومبیل را نمی‌رانید، شما از متدهای **faster** و **slower** (گاز و ترمز) **interface** راندن استفاده می‌کنید. شما کاری ندارید که متد **slower** (ترمز) چگونه پیاده‌سازی شده است. به عبارت دیگر اینکه ترمز ماشین هیدرولیک است یا ترمز هوا برای شما اهمیتی ندارد.

1-11- تصور کردن یک شیء :

در هنگام ایجاد یک شیء **COM**، خیلی نگران این هستید که **interface** ها چگونه کار می‌کنند. اما استفاده کننده شیء نگران پیاده‌سازی شما نیست. مثل ترمز اتومبیل، کاربر فقط مراقب است که **interface** کار کند اما نگران جزییات پشت **interface** نیست.

جدا کردن **interface** از پیاده‌سازی (**implemenation**) برای **COM** یک مسأله اساسی است. با جدا کردن **interface** از پیاده‌سازی‌اش، ما می‌توانیم شیء بسازیم. اشیاء قابل جانشینی و قابل استفاده مجدد هستند. این دو خاصیت مفید بودن اشیاء را چند برابر می‌کنند.

1-12- همراه با name چه چیزی مشخص می شود؟

حقیقت مهمی که باید بدانید این است که **interface** های دارای نامِ شیء **COM**، یکتا هستند. به همین خاطر، یک برنامه نویس اجازه دارد فرض کند که در استاندارد **C OM** اگر او به یک **interface** با نام مشخصی دسترسی پیدا کرد، تابع عضو **interface** و پارامترهایش دقیقاً در تمام اشیاء **COM** که آن **interface** را پیاده سازی کرده‌اند، مشابه است. بنابراین ، در ادامه مثالهایمان ، **interface** به نام « راندن» و « رادیو » در تمام اشیاء **COM** که آنها را فراهم می‌کنند، توابع کاملاً مشابهی دارند. اگر بخواهید هر تغییری در توابع عضو یک **i nterface** ایجاد کنید، باید یک **interface** جدید با یک نام جدید ایجاد کنید.

1-13- منبع تمام interface ها : IUnknown

توضیح قدیمی **COM** با توضیح دادن **IUnknown** شروع می‌شد. **IUnknown** پایه تمام **interface** های شیء **COM** است. برخلاف اهمیتش، لازم نیست برای دانستن مفاهیم **interface**، **IUnknown** را بدانید. **IUnknown** با لایه بالاتر تجریدی که ما در هنگام ایجاد شیء **COM** استفاده می‌کنیم، مخفی شده است. در واقع توجه بیش از حد به **IUnknown** گیج‌کننده است. پس اجازه بدهید به آن در اینجا به صورت سطح بالا نگاه کنیم تا به سراغ مفاهیم برویم. **IUnknown** مثل یک کلاس مجرد در **C++** است. تمام **interface** های **COM** باید از **Iunknown** به ارث برده شوند. **I Unknown** ایجاد و مدیریت **interface** ها را انجام می‌دهد. متدهای **I Unknown** برای ایجاد، شمارش **interface** ها و آزاد سازی اشیاء **COM** استفاده می‌شود. همه **interface** های **COM** این 3 متد را ایجاد می‌کنند و این 3 متد به‌صورت داخلی در شیء **COM** برای مدیریت **interface** ها استفاده می‌شود. شما تقریباً هیچگاه این 3 متد را خودتان صدا نمی‌زنید.

1-14- یک شیء COM معمولی

اکنون اجازه دهید تمام این مفاهیم جدید را کنار هم بگذاریم و یک شیء **COM** معمولی و برنامه‌ای که می‌خواهد به آن دست یابد را توضیح دهیم. در قسمتهای بعد کدی که برای پیاده‌سازی اشیاء **COM** لازم است را خواهیم آورد.

فرض کنید که شما می‌خواهید ساده‌ترین شیء **COM** معمولی را ایجاد کنید. این شیء یک **interface** را حمایت خواهد کرد و این **interface** فقط شامل یک **function** است. هدف این تابع نیز بسیار ساده است. این **function** بوق سیستم را به صدا درمی‌آورد. هنگامی که برنامه‌نویس این شیء **COM** را ایجاد کند و تابع عضو آن را در تنها **interface** ای که شیء حمایت می‌کند صدا بزند، بوق سیستم ماشینی که شیء **COM** روی آن قرار دارد به صدا درمی‌آید. اجازه دهید فرض کنیم که شما می‌خواهید این شیء **COM** را تنها روی یک ماشین اجرا کنید اما آن را از ماشین دیگری روی شبکه صدا بزنید.

چیزهایی که به آنها احتیاج دارید تا شیء **COM** را ایجاد کنید، عبارتند از:

- شما باید یک شیء **COM** ایجاد کنید و به آن یک نام بدهید. این شیء باید درون یک سرور **COM** پیاده‌سازی شود که از این شیء **COM** آگاه است.

- شما باید **interface** را تعریف کنید و به آن یک نام بدهید.
- شما باید توابعی داخل **interface** تعریف کنید و به آنها نام بدهید.
- در آخر باید **COM Server** را **install** نمایید.

برای این مثال اجازه دهید که شیء **COM** را **Beeper** بنامیم و **interface** آن را **IBeep** و تابع عضو آن را **Beep** نام دهیم. اولین مشکلی که در این نامگذاری فورا به ذهن می‌رسد آن است که در مدل **COM** همه ماشینها اجازه دارند که چندین **COM Server** را حمایت کنند که هر کدام شامل یک یا چند شیء **COM** است و هر شیء **COM** یک یا چند **interface** را حمایت می‌کند. این سرورها بوسیله برنامه‌نویسان متفاوتی ایجاد شده است و هیچ چیزی از انتخاب نام مشترک توسط برنامه‌نویسان جلوگیری نمی‌کند. به طریق مشابه، اشیاء **COM** دارای یک یا چند **interface** دارای نام خاص هستند که باز بوسیله برنامه‌نویسان مختلفی ایجاد شده‌اند که به طور تصادفی ممکن است نام مشابهی انتخاب کرده باشند. برای آنکه از برخورد (**collision**) نامها جلوگیری شود، کاری باید صورت گیرد. مفهوم **GUID** یا **Globally Unique IDentifier** مشکل «یکتا کردن نامها» را حل می‌کند.

1-15- چگونه یکتایی ایجاد می‌شود؟ (راه حل GUID)

در حقیقت دو راه شناخته شده برای اطمینان از یکنایی نامها وجود دارد:

1- نام‌ها را در یک سازمان رسمی به ثبت برسانیم.

2- از یک الگوریتم خاص استفاده کنیم که اعداد یکتایی ایجاد کند که یکتایی آنها در جهان قابل تضمین باشد(کار ساده‌ای نیست).

راه حل اول همان راه‌حلی است که برای **domain name** ها در شبکه استفاده می‌شود. این‌گونه برخورد مشکلش این است که برای ثبت هر نام جدید باید پولی بپردازید (مثلا **50$**) و چندین هفته به طول می‌انجامد تا این ثبت کردن تاثیر خود را بگذارد.

راه حل دوم برای توسعه دهندگان بسیار مناسب‌تر است. اگر شما بتوانید الگوریتمی اختراع کنید که تضمین کند که هر بار که کسی در سیاره زمین آن را فرامی‌خواند یک نام یکتا تولید کند، مساله حل شده است. در حقیقت، این مساله قبلا به وسیله **Open Software Foundation (OSF)** حل شده بود. **OSF** الگوریتمی ایجاد کرد که آدرس شبکه، زمان(که هر **100ns** یکی افزایش می‌یافت) و یک شمارنده را ترکیب کند. نتیجه یک عدد 128 بیتی خواهد بود که یکتا است.

عدد **2^128** یک عدد واقعا بزرگ است شما می‌توانید نانوثانیه‌های از ابتدای جهان تاکنون را با آن بشمارید و 39 بیت اضافه بیاورید. **OSF** این عدد را **UUID** (**Universal Unique**

IDentifier) نامید. مایکروسافت این الگوریتم را برای نامگذاری اشیاء **COM** استفاده کرد. در **COM** مایکروسافت معتقد بود که این عدد را نامگذاری مجدد کند و **GUID** بنامد. به طور مرسوم **GUID** را به شکل **Hex** می‌نویسند. مثلا:**"50709330-F9C0-11D0-BCE4-204C4F4F5020"**

از آنجا که استاندارد خاصی برای اعداد 128 بیتی در **C ++** وجود ندارد، از یک **structure** استفاده می‌کنیم. ساختار **GUID** شامل 4 فیلد است که احتمالاً هیچگاه احتیاجی به دستکاری آنها پیدا نمی‌کنید. این **structure** معمولاً به صورت زیر استفاده می‌شود:

```
typedef struct_GUID
{
unsigned long Data1;
unsigned short Data2;
unsigned short Data3;
unsigned char Data4[8];
} GUID;
```

GUID معمولاً به صورت **"gwid"** تلفظ می‌شود، اما برخی تلفظ زشت‌تر **"goo-wid"** را ترجیح می‌دهند. **GUID** به‌وسیله برنامه‌ای به نام **GUIDGEN** ایجاد می‌شود. در **GUIDGEN** شما یک کلید را فشار می‌دهید تا یک **GUID** جدید تولید شود. شما می‌توانید تضمین کنید تمام **GUID** هایی که ایجاد می‌کنید یکتا هستند؛ مهم نیست که چند **GUID** ایجاد می‌کنید و یا چند نفر در کره زمین **GUID** ایجاد می‌کنند. این روش درست کار می‌کند زیرا که می توانیم فرض زیر را در نظر بگیریم: تمام ماشینهای روی شبکه اینترنت (طبق تعریف) آدرس یکتا دارند. بنابراین ماشین شما باید در شبکه باشد تا **GUIDGEN** با قدرت واقعی خود کار کند. در واقع اگر شما یک آدرس شبکه نداشته باشید، **GUIDGEN** آن یک آدرس جعل می‌کند، اما احتمال یکتا بودن **GUID** پایین می‌آید.

هم اشیاء **COM** و هم **interface** های **COM** دارای **GUID** هستند تا آنها را مشخص کنند. بنابراین نام **Beeper** که برای شیئمان انتخاب کردیم در حقیقت نامربوط است. شیء با **GUID**اش نامگذاری می‌شود. ما **GUID** را **class ID** شیء می‌نامیم. ما می‌توانیم از یک **#define** یا **const** برای انتساب نام **Beeper** به **GUID** شیء استفاده کنیم تا عدد 128 بیتی مربوط در بین خطوط کد شناور نباشد. به طریق مشابه **interface** نیز دارای **GUID** است. توجه کنید که تعداد زیادی اشیاء **COM** که توسط تعداد زیادی برنامه‌نویس نوشته شده‌اند ممکن است از یک **interface** به نام **IBeep** حمایت کنند و همه آنها از یک **GUID** برای نامیدن آن استفاده کنند. اگر **GUID**ها یکسان نباشند، تا جایی که به شیء **COM** مربوط است آنها **interface** های مختلفی هستند. یعنی **GUID** نام است.

1-16- **COM server** :

COM server برنامه‌ای است که **interface** ها و کلاسهای شیء **COM** را ایجاد می‌کند.

COM server در 3 پیکر بندی زیر می‌تواند وجود داشته باشد :

1- **IN-process** یا **DLL server**

2- **stand-alone EXE server**

3- **windows NT based service**

اشیاء **COM** نیز صرفنظر از نوع **server** می‌توانند انواع بالا را داشته باشند. **interface** ها و **Coclass** های شیء **COM** نمی‌دانند که از چه نوع سروری استفاده می‌کنند. برای برنامه **client** نوع سرور اغلب کاملاً واضح است. نوشتن سرور، برای انواع مختلف آن کاملاً متفاوت است:

1- سرورهای **In-Process** به‌صورت **DLL** پیاده‌سازس می‌شوند. این شکل پیاده‌سازی به این معناست که سروربه طور دینامیک در زمان اجرا در **proces**،**load** می‌شود. **COM server** پس از **load** جزئی از برنامه می‌شود و اعمال شیء **COM** در داخل نحهای برنامه اجرا می‌شود. این روش، روش معمول پیاده‌سازی اشیاء **COM** است چرا که کارایی آن خارق‌العاده است (یک **overhead** مختصر برای فراخوانی توابع **COM** وجود دارد اما شما همه مزایای طراحی و استفاده مجدد را خواهید داشت). **COM** به طور اتوماتیک **load** شدن و **unload** شدن **dll** را کنترل می‌کند.

2- یک سرور **out-of-process** جداسازی بین **client** و **server** را بیشتر نمایان می‌کند. این‌گونه **server** ها به‌صورت برنامه اجرایی جداگانه اجرا می‌شود و بنابراین در یک فضای اجرایی پنهان از دسترس دیگر برنامه‌ها قرار دارند. شروع و خاتمه یک **server** به‌صورت **EXE**، به وسیله **windows service control manager(SCM)** کنترل می‌شود. فراخوانی **interface** های شیء **COM** به صورت مکانیزمهای ارتباط بین برنامه‌ای (**inter-process communication mechanisms**) انجام می‌شود. سرور می‌تواند روی یک کامپیوتر محلی اجرا شود یا روی یک کامپیوتر راه دور. اگر سرور روی یک کامپیوتر راه دور باشد، ما به آن با نام **Distributed COM** یا **DCOM** اشاره می‌کنیم.

3- **Windows NT** مفهوم **service** را عرضه کرده است. **service** برنامه‌ای است که به طور اتوماتیک به وسیله **Windows NT** مدیریت می‌شود و با کاربر کامپیوتر سرور کار ندارد. یعنی سرویس می‌تواند در زمان بوت شدن سیستم به طور اتوماتیک شروع می‌شود و در حال اجرا باشد حتی اگر هیچ کاربری به **Windows NT**، **logon** نکرده باشد. **service** ها راه بسیار مناسبی برای اجرای برنامه های **COM server** عرضه می کنند.

4- نوع چهارمی برای **server** ها وجود دارد که **"Surrogate"** (قائم مقام) نامیده می شود. که اصولاً برنامه ای است که اجازه می‌دهد که یک سرور **in-process** به صورت از راه دور (**remote**) اجرا شود. **Surrogate** ها وقتی سرورهای **COM** را به صورت **dll base** روی شبکه ایجاد می‌کنید، مفید هستند.

17-1- Interaction (تعامل) بین client و server :

در استاندارد **COM**، برنامه **client** همه چیز را جلو می‌برد. سرور کاملاً **passive** است و فقط به درخواست‌ها پاسخ می‌دهد. یعنی سرور **COM** به شکل سنکرون در برابر فراخوانی‌های تابع از سوی **client** رفتار می‌کند.

- برنامه **client**، برنامه سرور را راه اندازی می‌کند.
- برنامه **client**، شیء **COM** و **interface** هایش را تقاضا می‌نماید.
- برنامه **client**، تمام فراخوانی توابع از سرور را انجام می‌دهد.
- برنامه **client**، **interface** های سرور را آزاد می‌کند و اجازه می‌دهد سرور **shutdown** شود.

این امتیاز مهم است. راههایی برای شبیه سازی ارتباط به این صورت که فراخوانی از سرور به سمت **client** باشد، وجود دارد اما آنها برای پیاده سازی مشکل هستند و به شدت پیچیده‌اند.(آنها **call back** نامیده می‌شوند). به طور معمول سرور هیچ کاری بدون تقاضای **client** انجام نمی‌دهد.
در جدول زیر تعامل‌های معمول بین **client** و سرور **com** آمده است.

پاسخ **server**	تقاضای **client**
1- اگر لازم باشد سرور را راه اندازی می‌کند؛ اگر سرور **In-process** باشد، **dll** لود می‌شود. سرورهای اجرایی به‌وسیله **SCM** اجرا می‌شوند.	
2- شیء **COM** تقاضا شده ایجاد می‌شود.	تقاضای دستیابی به **interface** خاصی از شیء **COM**، با تعیین کلاس و **interface** شیء **COM**
3- **interface** لازم به شیء **COM** ایجاد	

می‌شود. 4- شمارنده ارجاع به **interface** های شیء **COM** افزایش می‌یابد. 5- اشاره‌گری به **interface**، به **client** برگردانده می‌شود.	(به کمک **GUID**).
متد را روی شیء **COM** اجرا می‌کند.	فرا خوانی یک متد از **interface**
1- شمارنده ارجاع **interface** را کاهش می‌دهد. 2- اگر شمارنده صفر شده است، شیء **COM** را **delete** می‌کند. 3- اگر **connection** فعال دیگری نباشد، سرور را **shut down** می‌کند. برخی از سرورها خودشان را **shut down** نمی‌کنند.	آزاد سازی **interface**

اگر می‌خواهید مدل **COM** را بفهمید، باید دیدگاه مرکزیت **client** داشته باشید.

1-18- **خلاصه** :

سعی کردیم به مدل **COM** از دیدگاههای مختلفی بنگریم. **C++** زبان اصلی **COM** است. اما خیلی مهم است که از دیدگاه شباهتها به آنها نگاه کنیم. مدل **COM** شباهتهای زیادی با **C++** دارد اما تفاوتهای مهمی نیز با آن دارد. مدل **COM** یک راه کاملاً جدید برای ارتباط بین **client** و سرور پیشنهاد می‌کند.

interface یکی از مهمترین مفاهیم مدل **COM** است. تمام تعاملهای مدل **COM** از طریق **interface** ها انجام می‌شود و آنها تعاملها را شکل می‌دهند. از آنجا که **interface** ها همتایی در **C++** ندارند، درک آنها برای مردم کمی دشوار است. همچنین مفهوم **GUID** مطرح شد. **GUID** در مدل **COM** خیلی مهم است و راه خوبی برای تشخیص موجودیتها در یک شبکه بزرگ را عرضه می‌کند

سرورهای **COM** فقط حامل اشیاء **COM** هستند. همه چیز بر پاسخگویی اشیاء **COM** به برنامه‌های **client** متمرکز شده است. در قسمت بعد یک **client** و سرور ساده ایجاد می‌کنیم.

2- ساده ترین شیء COM client :

مستقیم ترین راه برای درک **COM** نگاه کردن به آن از دیدگاه برنامه **client** است. چرا که هدف برنامه نویسی با اشیاء **COM** ایجاد اشیاء مفید برای برنامه‌های **client** است. وقتی **client** را فهمیدید، آنگاه فهمیدن سرور بسیار آسان‌تر می‌شود. جدا نگاه کردن به **c lient** و **server** ممکن است گیج کننده باشد. در ابتدا اجازه دهید با ساده‌ترین تعریف شروع کنیم: **C OM client** برنامه‌ای است که از مدل **COM** برای صدا زدن توابع **COM server** استفاده می‌کند. واضح ترین مثال برای این ارتباط **client/server** یک برنامه **User Interface** است **(client)** که متدهای برنامه دیگری را صدا بزند **(server)** . اگر برنامه **User Interface** آن متدها را با استفاده از متد **COM** فراخوانی نماید، برنامه **User Interface** طبق تعریف، یک **COM client** خواهد بود.

ما این مطلب را به این دلیل به طور ساده توضیح دادیم که رابطه بین **C OM server** و **COM client** می‌تواند بسیار پیچیده‌تر از این باشد و معمولاً اینگونه است. در بسیاری موارد ، برنامه **client**، **COM server** می‌شود و برنامه سرور **COM client** خواهد بود. کاملاً معمول است که یک برنامه هم **COM client** باشد و هم **COM server** . در این بخش ما اشیاء **client** و **server** را کاملاً از هم جدا خواهیم کرد و با یک **c lient** خالص سروکار خواهیم داشت.

2-1- چهار گام برای اتصال

برنامه **Client** وقتی از مدل **COM** استفاده می‌کنید، برای اتصال به **Server** 4 گام اصلی طی می‌کند. البته اشیاء **Client** واقعی کارهای بیشتری انجام می‌دهند، اما اگر پوسته پیچیدگی‌های خود را کنار بگذارید، هسته اصلی همین 4 گام خواهد بود. در این قسمت ما شیء و **COM** را در پائین‌ترین سطح آن با استفاده از فراخوانی‌های ساده **C++** ایجاد می‌کنیم.

در اینجا خلاصه‌ای از کارهایی که باید انجام دهیم ذکر می‌شود:

1. **Initial** کردن زیر سیستم‌های **COM** و بستن آن در پایان کار
2. درخواست **COM** برای **Interface** مشخص روی **Server**
3. اجرای متدها روی **Interface**
4. آزاد کردن **Interface**

برای این مثال، ما یک **COM Server** بسیار ساده را در نظر می‌گیریم. ما فرض می‌کنیم این **Server** اکنون نوشته شده‌است و توضیح آن را برای قسمت بعد می‌گذاریم.

این **Server** یک **Interface** به نام **Ibeep** دارد که آن **Interface** تنها یک متد به نام **Beep** دارد.

Beep یک پارامتر برای مدت بوق زدن می‌گیرد. هدف ما در این بخش نوشتن ساده‌ترین **COM Client** ممکن است که به **Server** متصل شود و متد **Beep** را فراخوانی نماید.

کد **C++** زیر این 4 گام را پیاده‌سازی می‌کند. این کد یک کد واقعی قابل اجرا می‌باشد.

```
#include "..\BeepServer\BeepServer.h"
// GUIDS defined in the server
const IID IID_IBeepObj =
{0x89547ECD,0x36F1,0x11D2,{0x85,0xDA,0xD7,0x43,0xB2,0x32,0x69,0
x28}};
const CLSID CLSID_BeepObj =
{0x89547ECE,0x36F1,0x11D2,{0x85,0xDA,0xD7,0x43,0xB2,0x32,0x69,0
x28}};
int main(int argc, char* argv[])
{
        HRESULT hr;                             // COM error code
        IBeepObj *IBeep;                        // pointer to interface
        hr = CoInitialize(0);                   // initialize COM
        if (SUCCEEDED(hr))                     // macro to check for success
        {
                hr = CoCreateInstance(
                CLSID_BeepObj,                  // COM class id
                NULL,                           // outer unknown
                CLSCTX_INPROC_SERVER,           // server INFO
                IID_IBeepObj,                   // interface id
                (void**)&IBeep );               // pointer to interface
                if (SUCCEEDED(hr))
                {
                        hr = IBeep->Beep(800); // call method
                        hr = IBeep->Release();  // release interface
                }
        }
        CoUninitialize();                       // close COM
        return 0;
}
```

فایل **"BeepServer.h"** موقعی که سرور را کمپایل می‌کنیم، ساخته شده است.**BeepServer.h** یک **COM Server** به شکل **In_Process** است که در بخش بعد خواهیم نوشت.

چندین **header file** به طور خودکار توسط **Visual Studio** هنگام کمپایل کردن **Server** تولید می‌شوند. این **header file** های خاص، **IBeepObj Interface** را تعریف می‌نماید. کامپایل کردن کد **Server** همچنین **GUID** هایی تولید می‌کند که در بالای برنامه ما دیده می‌شود. ما فقط آنها را از پروژه **Server** به اینجا کپی کرده‌ایم.

اجازه دهید به هر کدام از 4 گام بالا با جزئیات بیشتر نگاهی بیاندازیم.

2-2- Initial کردن سیستم COM

این گام، گام ساده‌ای است. متدی که لازم داریم **( )CoInitialze** است که به صورت زیر آن را فراخوانی می‌کنیم:

```
CoInitialize(0);
```

این تابع یک پارامتر می‌گیرد و این پارامتر همیشه صفر است (این پارامتر از نسخه قبلی **OLE** باقی مانده است).

تابع **( )CoInitialize** کتابخانه **COM** را **Initial** می‌کند شما باید این تابع را قبل از انجام هر کاری فراخوانی نمائید. موقعی که برنامه پیچیده‌تری می‌نویسیم از نسخه توسعه یافته آن به نام **( )CoInitializeEx** استفاده می‌کنیم.

موقعی که کارتان با شیء و **COM** تمام شد، **( )CoUninitialize** را فراخوانی نمائید. این تابع کتابخانه **COM** را **de_allocate** می‌نماید. می‌توان این دو تابع را به ترتیب در توابع **( )InitInstance** و **( )ExitInstance** از کلاسهای **MFC** فراخوانی نمود.

بیشتر توابع **COM**، **error**ی به نام **HRESULT** برمی‌گردانند. این مقدار شامل چندین فیلد است که شدت، سهولت و نوع خطا را مشخص می‌کند. ما از ماکروی **SUCCEEDED**استفاده کرده‌ایم، زیرا **COM** می‌تواند چندین کد موفقیت مختلف برگرداند. چک کردن با کد موفقیت معمولی (**S_OK**) کار دقیقی نیست. بعدا **HRESULT** را بیشتر توضیح خواهیم داد.

2-3- تقاضای COM برای یک Interface خاص

چیزی که **com client**به دنبال آن می‌گردد، توابع مفیدی است که بتواند فراخوانی کند تا او را به اهدافش برساند. در مدل **com** شما به مجموعه‌ای از توابع مفید از طریق یک **interface** دسترسی خواهید داشت. **Interface** در ساده‌ترین شکل خود چیزی جز مجموعه‌ای از یک یا چند تابع مربوط به هم نیست. موقعی که ما یک **interface** از شیء **com** دریافت (**get**) می‌کنیم، در واقع یک اشاره‌گر به مجموعه‌ای از توابع را می‌گیریم.

برای دریافت اشاره‌گر به **interface** می‌توانید از تابع **( )CoCreateInstance** استفاده کنید. این تابع یک تابع بسیار قوی است که برای انجام کارهای زیر با زیر سیستم **COM** تعامل می‌کند:

- تعیین مکان **Server**

- راه اندازی، بار کردن یا اتصال به **Server**
- ایجاد یک شیء **COM** روی **Server**
- بازگرداندن یک اشاره‌گر به یک **Interface** به شیء **COM**

دو نوع داده مهم برای یافتن و دستیابی پیدا کردن به **Interface** ها وجود دارند: **CLSID** و **IID**. هر دو آنها **GUID** (**Globally Unique ID**) هستند.

GUID ها برای شناسائی یکتای همه کلاسها و **Interface** های **COM** استفاده می‌شود. برای گرفتن یک کلاس و **Interface** مشخص به **GUID** آنها احتیاج دارید. راههای زیادی برای گرفتن **GUID** وجود دارد. ما معمولا **CLSID** و **IID** را از **header file** ها در **Server** خواهیم گرفت. در مثال ما، **GUID** ها را با **#define** در ابتدای کار تعریف کرده‌ایم. این کار همچنین جستجوی **GUID**ها را با استفاده کردن از نام معمولی **Interface** ها ساده می‌کند.

تابعی که اشاره‌گر به **Interface** را در اختیار ما می‌گذارد **( ) CoCreateInstance** است.

```
hr = CoCreateInstance(
    CLSID_BeepObj,          // COM class id
    NULL,                   // outer unknown
    CLSCTX_INPROC_SERVER,   // server INFO
    IID_IBeepObj,           // interface id
    (void**)&IBeep );       // pointer to interface
```

پارامتر اول یک **GUID** است که به طور یکتا کلاس شیء **COM** که **Client** می‌خواهد استفاده کند، را می‌شناساند. این **GUID** یک **COM Class identifier** یا **CLSID** است. هر کلاس **COM** در کره زمین **CLSID** یکتای خودش را دارد. **COM** از این **ID** برای تعیین اتوماتیک **Server** ی که بتواند شیء **COM** تقاضا شده را ایجاد نماید، استفاده می‌کند. وقتی که اتصال با **Server** برقرار شد، **Server** شیء خواسته شده را ایجاد می‌کند.

پارامتر دوم اشاره‌گر به چیزی است که **" outer unknown "** نامیده می‌شود. ما از این پارامتر استفاده نمی‌کنیم، بنابرین ما **NULL** به جای آن ارسال کرده‌ایم. **" outer unknown "** وقتی مفهوم **aggregation** (تجمع) را بررسی می‌کنیم، مهم خواهد بود.

Aggregation این امکان را فراهم می‌آورد که یک **Interface** به طور مستقیم **Interface** دیگری را بدون اینکه **client** از این طریق با خبر شود، فراخوانی نماید. **Aggregation** و **Coatainment** (محدود سازی) دو روشی است که **Interface** ها برای فراخوانی **Interface** های دیگر استفاده می‌کنند.

پارامتر سوم **COM Class Content** یا **CLSCTX** را تعیین می‌کند. این پارامتر میدان دید (**scope**) **Server** را کنترل می‌کند. بر اساس مقدار این پارامتر، کنترل می‌کنیم که **Server** یک

In_Process Server یا **EXE** یا روی یک کامپیوتر راه دور باشد. **CLSCTX** به صورت **bit_mask** است، بنابرین مقادیر مختلفی را می‌توانید در آن ترکیب کنید. ما از **CLSCTX_INPROC_SERVER** استفاده کرده‌ایم که بدین معناست که **Server** روی کامپیوتر محلی اجرا می‌شود و به **Client** به شکل یک **DLL** متصل می‌گردد. ما **IN_PROCESS SERVER** را برای این مثال انتخاب کرده‌ایم، زیرا پیاده‌سازی آن ساده‌تر است.

معمولاً **Client** نگران آن نیست که **Server** چگونه پیاده‌سازی شده‌است. در این حالت **Client** از مقدار **CLSCTX_SERVER** استفاده خواهد کرد که هر کدام از **IN_PROCESS SERVER** یا **Local** را که آماده باشد به کار خواهد گرفت.

پارامتر چهارم **Interface Identifire** یا **IID** است. این یک **GUID** دیگر است که این بار **Interface** ی را که تقاضا داریم مشخص می‌کند. **IID** ای که تقاضا می‌کنیم باید **IID** مربوط به **Interface** ای باشد که شیء **COM** حمایت می‌کند. این حمایت کردن در **COM Class** مشخص می‌شود و خود **COM Class** با یک **CLSID** شناخته می‌شود. باز هم مقدار **IID** معمولا بوسیله یک **header file** یا جستجوی نام **Interface** در برنامه **Server** ایجاد می‌شود.

آخرین پارامتر اشاره‌گر به یک **Interface** است. تابع **CoCreateInstance( )** ، **Class** شیء تقاضا شده و **Interface** آن را ایجاد می‌کند و اشاره‌گری به **Interface** را برمی‌گرداند. این پارامتر تنها دلیل فراخوانی **CoCreateInstance( )** است. پس از ان، می‌توانیم از اشاره‌گر به **Interface** برای فراخوانی متدهای روی **Server** استفاده می‌کنیم.

2-4- اجرای یک متد روی interface

CoCreateInstance از مدل **COM** استفاده می‌کند تا یک اشاره‌گر به **interface** ، **IBeep** ایجاد نماید. می‌توانیم این‌طور در نظر بگیریم که این اشاره‌گر، اشاره‌گری به یک کلاس معمولی **C++** است، اما در واقع این طور نیست. در واقع اشاره‌گر **interface** به یک ساختار به نام **VTABLE** اشاره می‌کند که جدولی از آدرسهای توابع است. می‌توانیم از عملگر **->** برای دستیابی به اشاره‌گر **interface** استفاده کنیم.

از آنجا که مثال ما از یک سرور **In_Process** استفاده می‌کند، این سرور به صورت یک **dll** در فرآیند برنامه ما لود خواهد شد. جدا از جزییاتی که در مورد **interface** وجود دارد، تمام هدف ما از از گرفتن این **interface**، فراخوانی یک متد روی سرور بوده است.

```
hr = IBeep -> Beep (800) ;
```

تابع **Beep** روی سرور اجرا می‌شود و باعث می‌شود کامپیوتر بوق بزند. راههای ساده‌تری نیز برای بوق زدن کامپیوتر وجود دارد. اگر یک سرور راه دور داشته باشیم که روی کامپیوتر دیگری اجرا می‌شود، آن کامپیوتر بوق خواهد زد.

متدهای یک **interface** معمولا پارامتر دارند. این پارامترها باید یکی از انواعی باشد که مدل **COM** اجازه می‌دهدئ قوانین زیادی مجاز بودن پارامتر برای **interface** را کنترل می‌کنند که ما بعدا آنها را در بحث **MIDL** (که ابزار تعریف **interface** برای اشیاء **COM** است)، توضیح خواهیم داد.

2-5- آزاد کردن interface

یک قاعده کلی در برنامه **C++** آن است که هر چیزی که **allocate** می‌شود، باید **deallocate** گردد. از آنجا که **interface** را با **new** ایجاد نکرده‌ایم، نمی‌توانیم با **delete** آن را حذف نماییم. تمام **COM interface** ها متدی به نام **Release** دارند که اتصال شیء را از بین می‌برد (**disconnect**) و آن را حذف می‌نماید (**delete**). رها کردن **interface** از آن جهت مهم است که به سرور اجازه می‌دهد خود را پاک‌سازی نماید (**Clean up**). اگر **interface** را با **CoCreateInstance** ایجاد کرده باشید، لازم است **Release** را فراخوانی نمایید.

2-6- خلاصه

در این قصل به یک **COM Client** ساده نگاهی انداختیم. **COM** یک سیستم **client** گرا (**client driven**) است. همه چیز به این سو گرایش دارد که ایجاد یک **component** برای **client** به راحتی امکان‌پذیر باشد. شما احتمالا تحت تاثیر سادگی برنامه **client** قرار گرفته‌اید. 4 گامی که در اینجا معرفی شد به شما اجازه می‌دهد تعداد بسیار زیادی از **component** ها را در یک **range** وسیع از برنامه‌های مختلف استفاده کنید.

برخی از گامها مثل **CoInitialize** و **CoUninitialize** پایه‌ای هستند. برخی از گامهای دیگر در برخورد اول زیاد توجه شما را جلب نمی‌کنند. این مهم است که شما بدانید در برنامه‌نویسی سطح بالا همه چیزی که اتفاق می‌افتد، همان چیزهایی است که در این مثال اتفاق افتاد. جزییات این مثال با مثالهای بیشتر، واضح‌تر می‌گردد.

نسخه 5 و 6 از **Visual C++** برنامه **client** را با استفاده کردن از **smart pointer** ها و راهنمای کامپایلر **#import** ساده‌تر می‌کنند. ما این مثال را به شکل **C++** سطح پایین ارائه دادیم تا مفاهیم، بیشتر روشن شود.

در بخش بعد یک سرور ساده **in_process** خواهیم ساخت که **interface**، **IBeep** را مدیریت کند. ما در بخش بعد، به سراغ جزییات جالبی از **interface** ها و فعال‌سازی آنها خواهیم رفت.

3- درک یک DCOM Server ساده

تا اینجا به چگونگی استفاده از **COM** در یک برنامه **client** نگاه کردیم. برای **client**، مکانیزم برنامه‌نویسی **COM** بسیار ساده است. برنامه **client** از زیر سیستم **COM** یک **component** مشخص را می‌خواهد و آن شیء به طور معجزه آسایی به او تحویل داده می‌شود.

کد بسیار زیادی که برای همه کارهایی که برای مدیریت **component** ها انجام می‌شود لازم است، از دید **client** مخفی است. پیاده‌سازی واقعی یک شیء به هنر پیچیده آراستن **component** ها در کنار هم و استاندارد کردن **module** های برنامه‌ای احتیاج دارد. حتی اگر از **MFC** استفاده کنید، کار پیچیده است. بسیاری از تولیدکنندگان حرفه‌ای وقت زیادی برای تلاش در این مورد ندارند. همین که مدل **COM** انتشار داده شد، به سرعت معلوم شد که عملی نیست که توسعه‌دهندگان کدها را خودشان بنویسند.

وقتی که به کد واقعی که برای پیاده‌سازی **COM** احتیاج است نگاه می‌اندازید، متوجه می‌شوید که مقدار زیادی از آنها تکراری هستند. برخورد سنتی **C++** با این‌گونه مشکلات، ایجاد یک کتابخانه کلاس است و در واقع **MFC OLE class** ها بسیاری از خواص **COM** را فراهم می‌کنند.

اما دلایل چندی وجود دارد که **MFC** و **OLE** انتخابهای خوبی برای **COM component** ها نیستند. با شروع بحث **ActiveX** و استراتژی اینترنتی مایکروسافت، برای اشیاء **COM** جمع و جور بودن و سرعت زیاد اهمیت پیدا کرد. در **ActiveX** احتیاج است که شیء **COM** در شبکه بسیار سریع کپی شود. اگر با **MFC** زیاد کار کرده باشید می‌دانید که هر چیزی می‌توان از آن انتظار داشت بجز

جمع و جور بودن (مخصوصا اگر برنامه به صورت استاتیک **link** شود). انتقال کد حجیم اشیاء تولید شده بوسیله **MFC** روی شبکه کار غیرممکنی است.

شاید بزرگترین مشکل دیدگاه **MFC/OLE** به **component** های **COM**، پیچیدگی این دیدگاه باشد. برنامه‌نویسی **OLE** مشکل است و بیشتر برنامه‌نویسان با آن بیگانه نیستند. تعداد زیاد کتابها، درباره **OLE** این حقیقت را نشان می‌دهد که استفاده از آن مشکل است.

بخاطر مشکلاتی که برای توسعه **OLE** وجود دارد، مایکروسافت ابزار جدیدی به نام **ATL** (**Active Template Library**) ایجاد کرده است. برای برنامه‌نویسی مدل **COM** ، **ATL** قطعا عملی‌ترین ابزار حاضر برای استفاده است. در حقیقت استفاده از **ATL Wizard** نوشتن **COM Server** را (اگر به دانستن آنچه در پشت صحنه می‌گذارد غلاقمند نباشید) بسیار آسان می‌کند.

مثالی که در اینجا می‌آوریم با استفاده از **ATL** و **ATL Wizard** است. این فصل چگونگی ایجاد یک سرور را برمبنای **ATL** توضیح می‌دهد و توضیح خلاصه‌ای از کدی که **wizard** تولید می‌کند، ارائه می‌دهد.

یکی از چیزهایی که درباره نوشتن سرور با **ATL** باید بدانید، این است که آنها شبیه برنامه‌های سنتی نیستند. یک **COM Server** در واقع حاصل همکاری چند جزء جداگانه است:

- برنامه شما
- زیر سیستم **COM**
- **ATL Template classes**
- کد **IDL** و برنامه‌ها و **header file** های تولید شده توسط **MIDL**
- **registry** ی سیستم

نگاه کردن به یک برنامه **COM** نوشته شده بر مبنای **ATL** و دیدن آن به شکل تماما متحد ممکن است مشکل باشد. حتی وقتی می‌دانید که آن شیء چه کاری می‌کند، باز هم مقدار قابل توجهی از آن را نمی‌توانید ببینید. مقداری از منطق واقعی سرور عمیقا در **header file** های **ATL** مخفی شده است. شما یک تابع **main** تنها که سرور را مدیریت و کنترل کند، نخواهید یافت. چیزی که خواهید یافت یک پوسته (**shell**) نازک است که اشیاء استاندارد **ATL** را فراخوانی می‌کند.

در قسمت بعد ما می‌خواهیم همه قسمتهایی که لازم است تا **server** اجرا شود را در کنار هم بگذاریم. در ابتدا، سرور را با استفاده از **ATL COM AppWizard** ایجاد خواهیم کرد. گام دوم افزودن یک شیء **COM** و یک متد است. ما یک سرور **In_process** می‌نویسیم زیرا یکی از ساده‌ترین سرورهای **COM** در پیاده‌سازی است. سرور **In_process** همچنین احتیاجی به ایجاد شیء **proxy** و **stub** ندارد.

3-1- ایجاد یک سرور COM برمبنای dll (سرور In_process)

یک سرور **In_process** یک کتابخانه مدل **COM** است که در زمان اجرا در برنامه شما لود می‌شود. به غبارت دیگر یک شیء **COM** در یک **dll** است. در واقع **dll** در نگاه سنتی سرور نیست، چرا که به طور مستقیم در فضای آدرس **client** لود می‌شود. اگر با **dll** ها آشنا باشید، چیزهای زیادی در مورد اینکه چگونه اشیاء **COM** لود می‌شوند و در برنامه صدا کننده **map** می گردند می‌دانید.

معمولا **dll** وقتی لود می‌شود که **LoadLibrary** فراخوانی شود. در **COM**، هیچگاه به وضوح **LoadLibrary** را صدا نمی‌زنید. وقتی برنامه **client**، **CoCreateInstance** را صدا می‌زند، همه چیز اتوماتیک شروع می‌شود. یکی از پارامترهای **CoCreateInstance** ، **GUID** کلاس شیء **COM** مورد درخواست شماست. هنگامی که سرور ایجاد می‌شود، در زمان کامپایل خود، تمام اشیاء **COM** را که حمایت می کند، ثبت (**register**) می‌نماید. وقتی **client** به شیء سرور احتیاج دارد، زیرسیستم **COM** محل **dll** سرور را پیدا می کند و به طور اتوماتیک آن را لود می‌نماید. پس از لود شدن، **dll** با استفاده از **Class factory** که در آن تعریف شده است شیء **COM** را ایجاد می کند.

CoCreateInstance اشاره‌گری به شیء **COM** برمی گرداند که به نوبه خود برای صدا کردن متدها استفاده می‌شود (در مثالی که توضیح داده شد این متد **Beep** بود). یک خاصیت خوب **COM** این است که **dll** می‌تواند وقتی مورد نیاز نیست به طور اتوماتیک **unload** شود. وقتی که **CoUninitialize** فراخوانی شد و شیء آزاد شد، **FreeLibrary** برای **unload** کردن **dll** سرور فراخوانی خواهد شد.

دانستن دقیق همه مطالب بالا لازم نیست. در واقع برای استفاده از مدل **COM** لازم نیست چیزی درباره **dll** ها بدانید. همه کاری که باید بکنید فراخوانی **CoCreateInstance** است. یکی از مزایای مدل **COM** آن است که جزییات را می‌پوشاند و بنابراین لازم نیست درباره آنها نگران باشید.

سرور **COM** به شکل **In_process** مزایا و معایبی دارد. اگر پیوند پویا بخش مهمی از طراحی سیستم شما باشد، درخواهید یافت که **COM** راه مناسبی برای مدیریت **dll** ها فراهم می‌آورد. برخی از برنامه‌نویسان باتجربه همه **dll** های مورد نیاز خود را به شکل **In_process COM Server** می‌نویسند. **COM** تمام مشکلات مربوط به **load** ، **unload** و **export** کردن **dll** ها را بر عهده می گیرد و در عوض فراخوانی توابع **COM** کمی **overhead** اضافی خواهد داشت.

اما دلیل اصلی ما برای انتخاب یک سرور **In_process** این بوده است که مثال ما را ساده‌تر می کند. در این حالت لازم نیست درباره شروع به اجرای سرور راه دور (به صورت **Exe** یا **Service**)

نگران باشیم چرا که سرور وقتی که مورد نیاز باشد به طور اتوماتیک لود می‌شود. همچنین لازم نیست **dll** مربوط به **proxy/stub** را برای انجام **marshalling** بسازیم.

بدبختانه، به خاطر آنکه سرور **In_process** بسیار محدود به **client** ما است، تعدادی از مفاهیم مهم توزیع شدگی (**distributed**) مدل **COM** تحقق نمی‌یابد. سرور **dll** حافظه را با **client** اش به اشتراک می‌گذارد، در حالی که یک سرور **distributed** بیشتر از این باید از **client** اش جدا باشد. فرآیند انتقال داده بین **client** و **server** توزیع شده ، **marshalling** گفته می‌شود. **marshalling** محدودیتهای مهمی بر تواناییهای **COM** تحمیل می‌کند که در سرور **In_process** نباید درباره آن نگران باشیم.

3-2- ایجاد سرور با استفاده از ATL Wizard

ما می‌خواهیم در این مثال یک سرور **COM** خیلی ساده ایجاد کنیم به طوری که در هم ریختگی‌های یک سرور تمام عیار را نادیده بگیریم و به شما کمک کنیم که مفاهیم پایه‌ای که در پشت **COM** است را به سرعت فرابگیرید. سرور تنها یک متد دارد (**Beep**). تمام کاری که این متد انجام می‌دهد آن است که یک صدای بوق ایجاد می‌کند. کاری که ما می‌خواهیم انجام دهیم **setup** کردن تمام قسمتهای یک سرور واقعی است. وقتی معماری کلی ایجاد شد، افزودن متدهایی برای انجام کارهای مفید کاملا واضح خواهد بود.

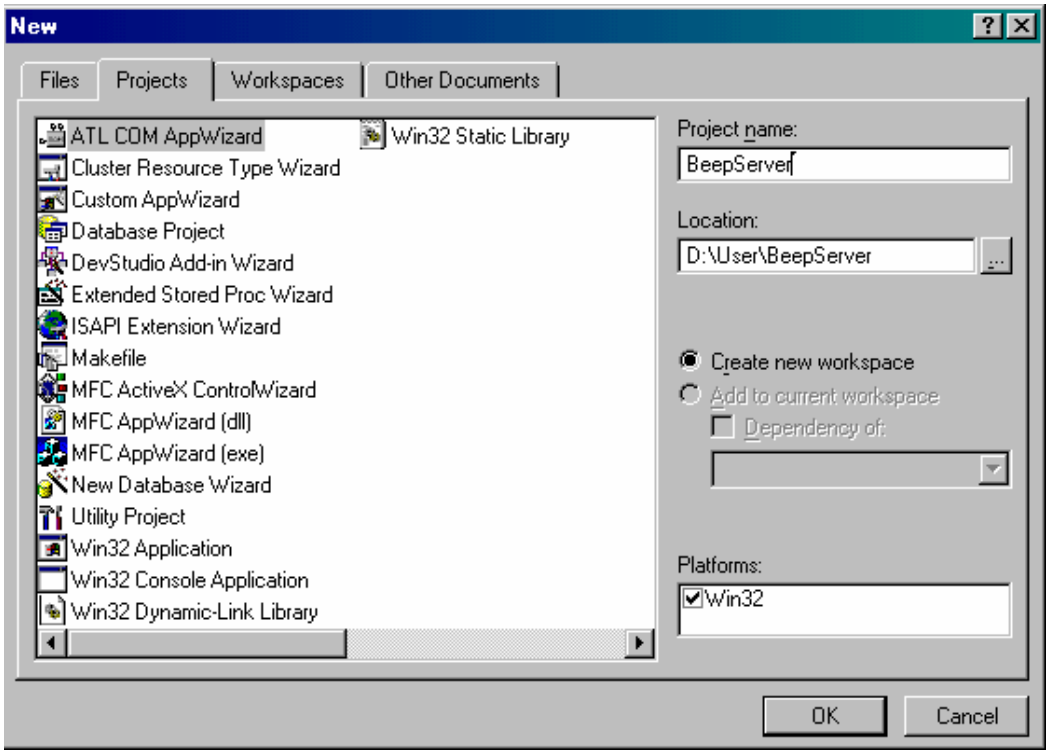
ATL AppWizard راه آسانی برای تولید سریع **COM Server** است. **wizard** به ما اجازه می‌دهد تمام **option** های پایه‌ای را انتخاب کنیم و بیشتر کد مورد نیاز ما را تولید می‌کند.

در زیر گامهای لازم برای ایجاد سرور به صورت قدم به قدم می‌آید. در این مثال نام سرور را **BeepServer** گذاشته‌ایم. هر **COM Server** باید حداقل یک **interface** داشته باشد و **interface** این مثال **IBeepObject** نامیده می‌شود. شما می‌توانید **interface** های شیء **COM** خود را تقریبا هر چیزی که می‌خواهید نامگذاری کنید اما اگر می‌خواهید استاندارد نامگذاری مرسوم را رعایت کنید، همیشه باید پیشوند آنها را **I** بگذارید.

توجه: در بیشتر **document** های شرکت مایکروسافت برای تمایز قائل شدن بین کلاس **COM** و کلاس **C++** معمولی، برای کلاس **COM** از کلمه **Coclass** استفاده می‌شود.

گامهای ایجاد **COM Server** جدید با استفاده از **ATL Wizard** در نسخه 6 **Visual C++** به شکل زیر می‌باشد:

1. ابتدا با انتخاب **New** از منوی **File** یک پروژه **ATL COM AppWizard** ایجاد می‌کتیم.

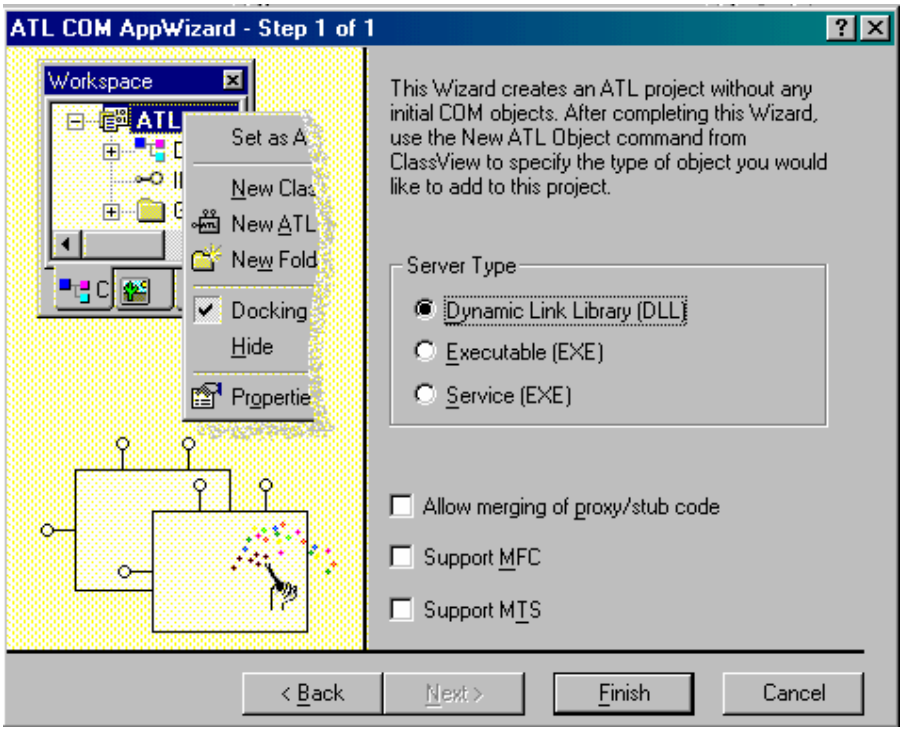


2. در پنجره **New** انتخابهای زیر را انجام می‌دهیم و کلید **Ok** را فشاریم.

• Project Name: BeepServer
• Create New Workspace
• Location: Your working directory

3. در اولین Dialog Box یک DLL Base Server ایجاد می‌کنیم. تنظیمات زیر را وارد کنید.

• Dynamic Link Library
• Don't allow merging proxy/stub code
• Don't support MFC



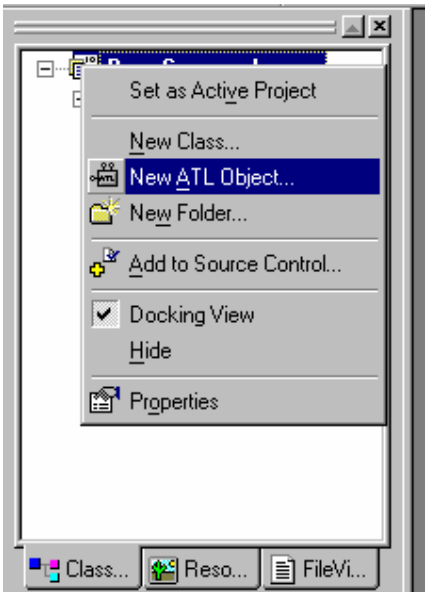
4. کلید **Finish** را بفشارید.

AppWizard یک پروژه **DLL Base COM Server** با تمام فایلهای مورد نیاز ایجاد می‌کند. با وجود اینکه این **Server** کامپایل و اجرا می‌شود، تنها یک پوسته خالی است. و برای اینکه مفید باشد به یک **Interface** نیاز داریم. همچنین برای این **Interface** باید متد بنویسیم.

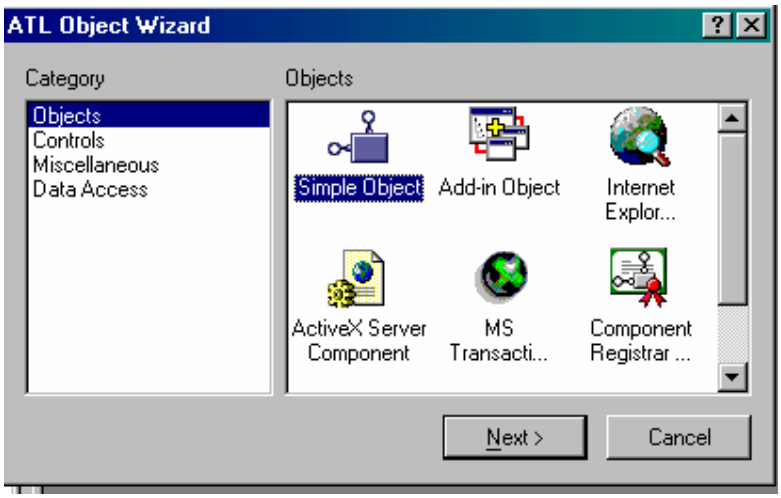
3-3- افزودن یک شیء COM و یک متد

اکنون می‌خواهیم یک شیء **COM** و یک **Interface** با تعدادی متد برای آن بنویسیم.

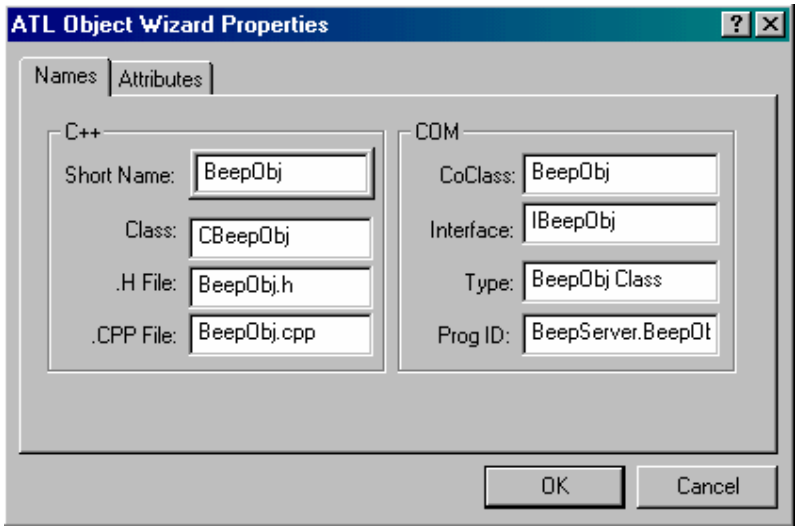
1- در پنجره **Class View** بر روی **BeepServer Classes** کلیک راست کنید و **New ATL Object**را انتخاب کنید.



2- در پنجره **Object Wizard** گروه را **Objects** و شیء را **Simple Object** انتخاب کنید و کلید **Next** را انتخاب کنید.

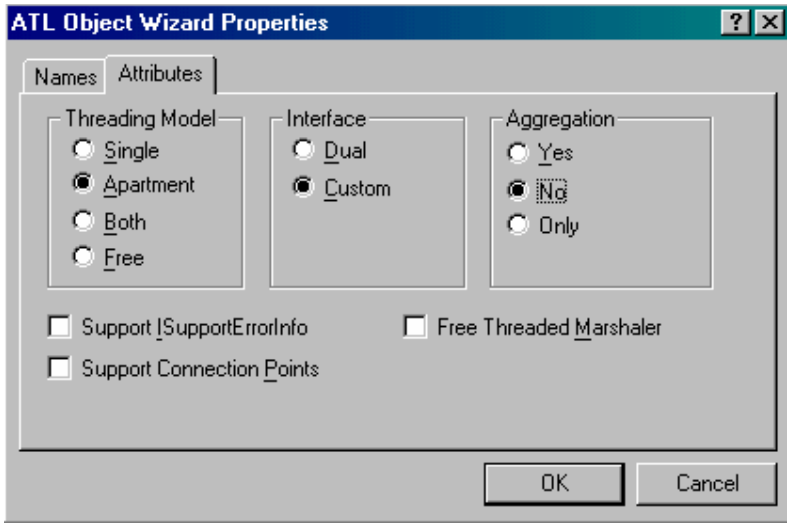


3- در برگه **Names**، **Short Name** را **BeepObj** انتخاب کنید بقیه فیلدها به صورت اتوماتیک پر می‌شوند.



4- در برگه **Attributes** انتخابهای زیر را انجام دهید.

Apartment Threading
Custom Interface
No Aggregation

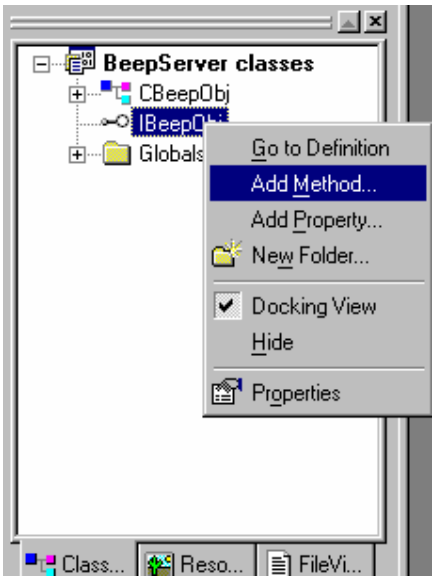


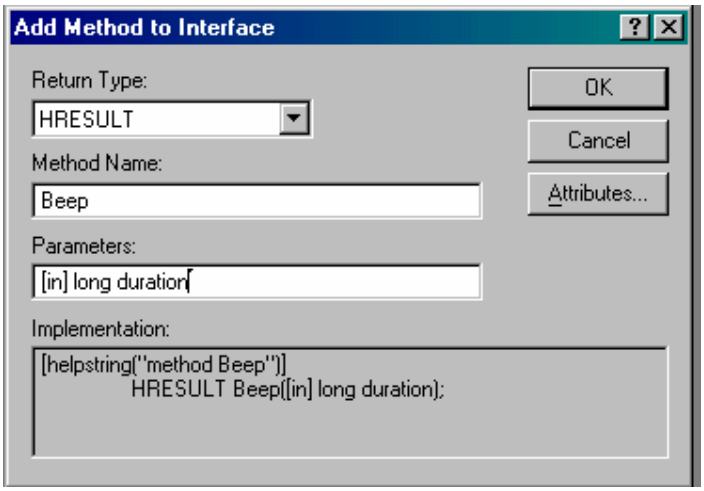
5-کلید **OK** را فشار دهید تا شیء **COM** ایجاد شود.

3-4- اضافه کردن یک متد به Server

تا کنون یک شیء **COM** خالی ساخته‌ایم، بنابرین هنوز قابل استفاده نشده است. ما یک متد ساده برای آن می‌نویسیم تا تابع **Beep( )** را صدا بزند. تابع **Beep( )** باعث می‌شود کامپیوتر یکبار بوق بزند.

1. به برگه **Class View** بروید و **IBeepObj interface** را انتخاب کنید.
2. روی **IBeepObj** کلیک راست کرده گزینه **Add Method** را انتخاب کنید.





3. پنجره ظاهر شده را مطابق شکل زیر کامل کنید.

4. حال می‌خواهیم برای متد ایجاد شده برنامه‌ای بنویسیم. فایل **BeepObj.CPP** را باز کرده، در قسمت **//TODO** یک خط اضافه کنید و تابع **Beep** را فراخوانی نمایید.

```
STDMETHODIMP CBeepObj::Beep(LONG duration)
{
// TODO: Add your implementation code here
::Beep( 550, duration );
return S_OK;
}
```

5. فایل را ذخیره کرده و پروژه را کامپایل کنید.

اجازه دهید که به فایلهایی که **Wizard** برای ما ایجاد کرده است، نگاهی بیاندازیم. جدول زیر توضیح مختصری راجع به اکثر آنها می‌دهد.

Source File	Description
BeepServer.dsw	Project workspace
BeepServer.dsp	Project File
BeepServer.plg	Project log file. Contains detailed error information about project build.
BeepServer.cpp	DLL Main routines. Implementation of DLL Exports
BeepServer.h	MIDL generated file containing the definitions for the interfaces
BeepServer.def	Declares the standard DLL module parameters: DllCanUnload-Now, DllGetClassObject, DllUnregisterServer
BeepServer.idl	IDL source for BeepServer.dll. The IDL files define all the COM components.
BeepServer.rc	Resource file. The main resource here is IDR_BEEPDLLOBJ which defines the registry scripts used to load COM information into the registry.
Resource.h	Microsoft Developer Studio generated include file.
StdAfx.cpp	Source for precompiled header.
Stdafx.h	Standard header
BeepServer.tlb	Type Library generated by MIDL. This file is a binary description of COM interfaces and objects. The TypeLib is very useful as an alternative method of connecting a client.
BeepObj.cpp	Implementation of CBeepObj. This file contains all the actual C++ code for the methods in the COM BeepObj object.
BeepObj.h	Definition of BeepObj COM object.
BeepObj.rgs	Registry script used to register COM components in registry. Registration is automatic when the server project is built.
BeepServer_i.c	Contains the actual definitions of the IID's and CLSID's. This file is often included in cpp code.
	There are several other proxy/stub files that are generated by MIDL.

5-3- خلاصه

تقریبا تمام کد سرور به وسیله **ATL Wizard** ایجاد می‌شود. مثال ما درباره سرور **In_Process** بود ولی روش کار برای همه انواع سرورها یکی است. این ابزار، ابزاری بسیار مناسب برای توسعه سریع برنامه سرور است چرا که نیاز نیست جزییات دقیق مورد نیاز را بدانید.

4- پیاده‌سازی یک Client و Server به صورت توزیع شده

بیشتر برنامه‌نویسان **COM** فقط از سرور **in_process** به صورت **local** استفاده می‌کنند که به صورت **dll** اجرا می‌شود. **Dll** در فضای آدرس برنامه **client** لود می‌شود و بنابراین سروری که به این صورت نوشته می‌شود، مستحکم و کاراست. اما در این بخش ما می‌خواهیم سرور را به صورت فایل **Exe** پیاده‌سازی کنیم. این روش مشکلات جدیدی را ایجاد خواهد کرد.

یک خبر خوب آن است که تبدیل کردن از **COM** به **DCOM** آسان است و خبر بد آن است که خطایابی برنامه‌های **DCOM** مشکل است.

4-1- تفاوتهای بین COM و DCOM

بیشتر تفاوتهای برنامه‌های **COM** و **DCOM** از دید توسعه‌دهندگان مخفی است. این مفهوم به عنوان **Local/remote transparency** شناخته می‌شود. البته برخی تفاوتها در مورد نحوه کار داخلی **COM** و **DCOM** وجود دارد. ارتباطات محلی از راههای مختلفی مثل پیامهای **Windows** می‌تواند انجام شود. ارتباط با یک کامپیوتر راه دور احتیاج به یک لایه جدید از اشیا و ترافیک جدیدی روی شبکه دارد.

اینجا نیز شبیه **COM** همه چیز از **client** شروع می‌شود. در **DCOM** ، **client** تابع **CoCreateInstanceEx()** را فرامی‌خواند و توضیحات سرور و **CLSID** شیء مورد نظر را ارسال می‌کند و یک **interface** را تقاضا می‌کند. این تقاضا بوسیله **Service Control Manager (SCM)** که قسمتی از **Windows** است، رسیدگی می‌شود. **SCM** پاسخگوی ایجاد و فعال‌سازی شیء **COM** روی سرور است.

پس از ایجاد سرور راه‌دور **COM** ، تمام فراخوانی‌ها از طریق اشیاء **proxy** و **stub** انجام می‌شود. **proxy** و **stub** از طریق **RPC** رابطه برقرار می‌کنند و **RPC** تمام فعالیتهای شبکه‌ای را پاسخگوست. در **server** شیء **stub** منتظر **marshalling** است و شیء **proxy** در **client** این کار را انجام می‌دهد.

ارسال داده روی شبکه به کمک **RPc** انجام می‌شود. در واقع **DCOM** از نوع توسعه‌یافته **RPC** به نام **Object RPC** یا **ORPC** استفاده می‌کند. **RPC** روی تعدادی از پروتکلها شامل **TCP/IP** ،

UDP ، **NetBiue** و **NetBIOS** می‌تواند اجرا شود. در پروتکلی مثل **UDP** که **connection_less** است، نگه‌داشتن **connection** برعهده خود **DCOM** است.

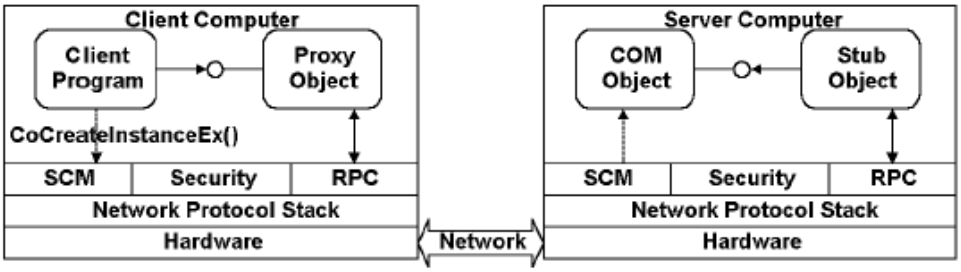


Figure 1. DCOM System Relationships

4-2- سرور تغییر زیادی نمی‌کند

سروری که به صورت یک برنامه اجرا شود (فایل **Exe**) روی شبکه کار می‌کند. بنابراین لازم نیست در سرور خود تغییری بدهید، مگر آنکه بخواهید برخی مسائل امنیتی را رعایت کنید.

اگر سرور به شکل **in_process** نوشته شده باشد، لازم است تغییراتی در برنامه‌تان ایجاد کنید. چرا که **dll** در فضای آدرس **client** لود می‌شود و نمی‌تواند تحت شبکه برای **client** را دور کار کند. راه حل آن ایجاد یک **surrogate** است که یک فایل اجرایی است که **dll** را به شکل یک پوشش در بر می‌گیرد. اما بهتر است برنامه خود را به صورت فایل **Exe** دوباره ایجاد کنید. برای این کار یک فایل **Exe** با **ATL Wizard** ایجاد کنید و کدهای خود را به آن منتقل کنید.

4-3- کد client

در زیر کد لازم برای یک برنامه **client** ساده برای **DCOM** می‌آید. تفاوت این برنامه با برنامه **client** قبلی این است که این **client** برخی از **error** ها را چک می‌کند و برخی مسائل امنیتی شبکه را چک می‌کند (**CoInitializeSecurity**) و **interface** را با تابع **CoCreateInstanceEx** دریافت می‌کند و به این تابع اطلاعات سرور را در ساختاری به نام **COSERVERINFO** ارسال می‌کند. توضیحات دیگر در هر خط برنامه آورده شده است.

```
// RemoteClient.cpp : Defines the entry point for the console application.
#include "stdafx.h"        // added _WIN32_DCOM
#include <iostream.h>    // get "cout"
#include <comdef.h>      // get _com_error
#include <time.h>        // get time_t

// extract definitions from server project
#include "..\RemoteServer\RemoteServer.h"
#include "..\RemoteServer\RemoteServer_i.c"

// forward reference for status display method
void ShowStatus( HRESULT hr );

int main(int argc, char* argv[])
{
        HRESULT hr; // COM error code
        IGetInfo *pI; // pointer to interface

        // Get the server name from user
        char name[32];
        cout << "Enter Server Name:" << endl;
        gets( name );
        _bstr_t Server = name;

        // remote server info
        COSERVERINFO cs;
        // Init structures to zero
        memset(&cs, 0, sizeof(cs));
        // Allocate the server name in the COSERVERINFO struct
        cs.pwszName = Server;

        // structure for CoCreateInstanceEx
        MULTI_QI qi[1];
        memset(qi, 0, sizeof(qi));

        // initialize COM
        hr = CoInitialize(0);
        ShowStatus( hr );

        // macro to check for success
        if (SUCCEEDED(hr))
        {
                // set a low level of security
                hr = CoInitializeSecurity(NULL, -1, NULL, NULL,
                                        RPC_C_AUTHN_LEVEL_NONE,
                                        RPC_C_IMP_LEVEL_IMPERSONATE,
                                        NULL,
                                        EOAC_NONE,
                                        NULL);

                // init security
                ShowStatus( hr );
```

```
}

if (SUCCEEDED(hr))
{
        // Fill the qi with a valid interface
        qi[0].pIID = &IID_IGetInfo;

        // get the interface pointer
        hr = CoCreateInstanceEx(
                        CLSID_GetInfo,      // clsid
                        NULL,               // outer unknown
                        CLSCTX_SERVER,      // server context
                        &cs,                // server info
                        1,                  // size of qi
                        qi );               // MULTI_QI array
        ShowStatus( hr );
}

if (SUCCEEDED(hr))
{
        BSTR bsName; // Basic style string

        // Extract the interface from the MULTI_QI strucure
        pI = (IGetInfo*)qi[0].pItf;

        // Call a method on the remote server
        hr = pI->GetComputerName( &bsName );
        ShowStatus( hr );

        // Convert name to a printable string
        _bstr_t bst( bsName );
        cout << "Server Name :" << bst << endl;

        // get time from remote computer
        time_t tt;
        hr = pI->GetTimeT(&tt );
        ShowStatus( hr );

        // display time_t as a string
        cout << "Server Time :" << ctime( &tt ) << endl;

        // Release the interface
        pI->Release();
}

// Close COM
CoUninitialize();

// Prompt user to continue
cout << "Press ENTER to continue" << endl;
getchar();
```

```
        return 0;
}
```

4-4- ثبت (register) کردن Server و Proxy/Stub

اگر روی یک ماشین کار می‌کنید **register** کردن **DCOm** مثل **register** کردن **COM** است. اما در غیر این صورت لازم است کارهایی انجام دهید.

برنامه‌هایی که با **ATL Wizard** ایجاد می‌شوند، وقتی با سوییچ **regserver–** آنها را اجرا کنید، خود را **register** می‌کنند و خارج می‌شوند. مثلا در مثال ما می‌نویسیم:

```
C\> remoteServer -regserver
```

در این مثال ما از **interface** های **custom** استفاده کرده‌ایم. بنابراین لازم است **dll** مربوط به اشیاء **proxy/stub** را نیز به کامپیوتری که می‌خواهیم سرور را روی آن نصب کنیم، کپی کنیم و آن را در ماشین مقصد **register** نماییم. در مثال ما دستوری به شکل زیر را می‌توانیم به کار ببریم:

```
C:/> REGSVR32 remoteserverps.dll
```

اگر می‌خواهید شیء **DCOM** را که به شکل **dual base** نوشته شده است ثبت نمایید، به جای ثبت **dll** مربوط به اشیاء **proxy/stub** لازم است **type library** مربوطه را **register** نمایید.

4-5- کپی و ثبت کردن client روی کامپیوتر دیگر

پس از آنکه **client** و **server** را روی یک کامپیوتر آزمایش کردید، احتمالا می‌خواهید **client** را در جای دیگری کپی کنید و آن را از راه دور آزمایش نمایید. برای این کار فایل اجرایی برنامه **client** و فایل **dll** مربوط به اشیاء **proxy/stub** (که همراه **server** تولید شده است) را به **client** کپی کنید. لازم نیست شیء **server** را روی ماشینی که قرار است برنامه **client** را اجرا کند **register** نمایید. برای مثال ما دستوراتی مثل دستورات زیر لازم است:

```
C:> COPY \\Raoul\UnderCOM\RempteClient\Debug\Remoteclient.EXE
C:> COPY \\Raoul\UnderCOM\RemoteServer\RemoteserverPS.DLL
C:> REGSVR32 RemoteserverPS.DLL
```

خط آخر را فراموش نکنید زیرا بدون آن هیچ چیز کار نمی‌کند.

مراجع

1-“Understanding DCOM”, ***William Rubin & Marshall Brain, 1999, Prentice Hall***

2-“Mastring COM and COM+”, ***Ash Rofail & Yasser Shohoud, 2000, Sybex***

3-“Distributed COM”, ***Jim Maloney, 1999, Prentice Hall***

4-“MSDN”, ***Microsoft, January 2000, Microsoft***