Klient COM
Smart Pointers

Ogólna struktura klienta COM

Aby klient mógł użyć serwera COM musi wykonać następujące kroki:

1. **Zainicjalizować system COM**
2. **Stworzyć instancję obiektu COM**
3. **Wywołać procedurę obiektu**

Klient COM
Inicjalizacja i deinicjalizacja systemu COM

Zanim wątek wykona jakiekolwiek wywołanie do serwera COM musi zainicjalizować system COM. Obecnie zaimplementowane są dwie funkcje inicjalizujące:

1. `CoInitialize()`
2. `CoInitializeEx()`

Klient COM
Inicjalizacja i deinicjalizacja systemu COM

Funkcja `CoInitialize()` inicjalizuje system COM z domyślnymi parametrami. `CoInitializeEx()` umożliwia zdefiniowanie sposobu uruchomienia serwera przez programistę. Funkcja ta przyjmuje 2 parametry. Pierwszy z nich przewidziany jest do przyszłych zastosowań i musi być NULL. Drugi z parametrów definiuje rodzaj współpracy z serwerem i jego optymalizację:

1. `COINIT_APARTMENTTHREADED`
2. `COINIT_MULTITHEREADED`
3. `COINIT_DISABLE_OLE1DDE`
4. `COINIT_SPEED_OVER_MEMORY`

Klient COM
Inicjalizacja i deinicjalizacja systemu COM

`COINIT_APARTMENTTHREADED`: synchronizacja klienta i serwera następuje poprzez kolejkę zdarzeń uruchomioną na osobnym wątku.

`COINIT_MULTITHEREADED`: zakłada, że serwer wspiera wielowątkowość i dostęp do obiektu następuje bezpośrednio bez udziału kolejki zdarzeń.

`COINIT_DISABLE_OLE1DDE`, `COINIT_SPEED_OVER_MEMORY`: wyłączają obsługę komunikatów DDE i optymalizują działanie systemu COM.

`CoInitialize(NULL)` jest równoważne wywołaniu `CoInitializeEx(NULL,COINIT_APARTMENTTHREADED)`.

Klient COM
Inicjalizacja i deinicjalizacja systemu COM

Niezależnie czy inicjalizację przeprowadzono przez `CoInitialize()` czy przez `CoInitializeEx()`, deinicjalizację systemu COM przeprowadza się przez `CoUninitialize()`.

Dla obiektów COM obsługujących schowek, technikę *drag and drop*, OLE1 lub OLE2, inicjalizację i deaktywację systemu COM należy przeprowadzić przez parę funkcji `OleInitialize()` i `OleUninitialize()`. Ponieważ funkcje OLE nie są *thread-safe*, inicjalizują one domyślnie system COM z parametrem `COINIT_APARTMENTTHREADED`.

Klient COM
Tworzenie instancji obiektu COM

Instancję obiektu można stworzyć poprzez omówioną wcześniej funkcję `CoCreateInstance()`. System COM pośrednio poprzez *class factory* tworzy pojedynczą instancję komponentu w zdefiniowany w rejestrze systemowym sposób.

Funkcja `CoCreateInstanceEx()` zapewnia możliwość autoryzacji, pozwala na stworzenie instancji obiektu w zdalnym systemie, i jednoczesne pobranie różnych wskaźników interfejsów obiektu.

Klient COM
Dostęp do deklaracji obiektu z poziomu kodu

Dostęp do deklaracji klas abstrakcyjnych obiektu serwera następuje poprzez:

- **Włączenie plików wygenerowanych przez kompilator IDL ("OBJ_i.c" i "OBJ.h")**
- **Dyrektywę `#import "FileName"`, gdzie FileName jest plikiem zawierającym bibliotekę typów (plik TLB, DLL, PKG)**

Dyrektywa `#import "FileName"` generuje pliki TLH i TLI zawierające deklarację interfejsu obiektu COM. Dodatkowo generuje ona inteligentne wskaźniki (*smart pointers*) dla obiektów znajdujących się w bibliotece.

Klient COM
Dostęp do deklaracji obiektu z poziomu kodu

Dyrektywa `#import "FileName"` może posiadać dodatkowe parametry, np.

```
#import "FileName" no_namespace
```

- **`no_namespace` określa, że biblioteka nie będzie włączona w oddzielny *namespace*. Domyślnym *namespace* jest zdefiniowana nazwa biblioteki typów.**
- **`rename_namespace` zmienia domyślną nazwę *namespace* na inną. Pozwala na uniknięcie ewentualnych konfliktów nazw.**
- **`no_implementation` powoduje wygenerowanie deklaracji (plik TLH) bez włączania implementacji (TLI) interfejsu. Wymaga aby w innej części projektu użyta została dyrektywa `#import` z parametrem `implementation_only`**

Klient COM
Dostęp do deklaracji obiektu z poziomu kodu

- `named_guids` generuje w pliku TLH stałe reprezentujące symbolicznie UUID komponentów i interfejsów (np.. IID_IStopwatch). Bez tego parametru dostęp do UUID możliwy jest przez makrodefinicję `__uuidof()` (np. `__uuidof(Stopwatch)`)
- `exclude("name1"[,"name2",...])` – pomija podczas generacji kodu w plikach TLI i TLH wskazane obiekty.
- `raw_interfaces_only` – generuje w plikach TLI i TLH jedynie kodu dla *smart pointers* bez obsługi properties za pomocą pseudoskładowych.

Klient COM
Inteligentny wskaźnik

Inteligentny wskaźnik obiektu COM zrealizowany jest przez tzw. *wrapper class*. Jest ona oparta na klasie-wzorcu `_com_ptr_t<>`. Podstawowymi cechami inteligentnego wskaźnika są:

- Wewnętrzna obsługa `CoCreateInstance()` w konstruktorze klasy.
- Wewnętrzna obsługa licznika odwołań
- Dostęp do wszystkich metod interfejsu. Dodatkowo wykorzystuje właściwość argumentu `retval` do zwracania argumentu przez wartość funkcji.
- Dostęp do właściwości poprzez "pseudoskładową".
- Obsługa operatorów kopiowania, konwersji, porównania.
- Udostępnia obsługę interfejsu w klasyczny sposób poprzez operator `->`
- Wykorzystuje wyjątki do zgłaszania błędów (`_com_error`).

Klient COM
Inteligentny wskaźnik

```
interface IStopwatch : IUnknown
{
        HRESULT Start();
        HRESULT ElapsedTime([out, retval] float* Time);
        [propput] HRESULT Overhead([in] float Time);
        [propget] HRESULT Overhead([out, retval] float* Time);
};
class IStopwatchPtr ...
{
...
        __declspec(property(get=GetOverhead, put=PutOverhead))
        float Overhead;

        HRESULT Start();
        float ElapsedTime();
        float GetOverhead();
        void PutOverhead(float Val);
...
};
```

Nazwa inteligentnego wskaźnika tworzona jest z nazwy interfejsu poprzez dodanie na końcu przyrostka `Ptr`.

Klient COM
Inteligentny wskaźnik

Inteligentny wskaźnik automatycznie zwiększa licznik odwołań przy operacjach konstrukcji i kopiowania z innego inteligentnego wskaźnika lub ze wskaźnika interfejsu.
Zmniejszanie licznika odwołań następuje poprzez przypisanie do inteligentnego wskaźnika NULL lub w jego destruktorze.
Należy zachować szczególną ostrożność przy jednoczesnym użyciu w kodzie inteligentnych wskaźników i interfejsów:

- operator konwersji do interfejsu nie wywołuje `AddRef()`
- po wywołaniu `Attach()` inteligentny wskaźnik przejmuje kontrolę nad interfejsem. W zależności od użytej funkcji `Attach()` dla wskaźnika interfejsu należy wywoływać `Release()` lub nie
- Po wywołaniu `Detach()` inteligentny wskaźnik przestaje być odpowiedzialny za kontrolę licznika odwołań

Klient COM
Klient Stopwatch – inteligentny wskaźnik

```
#import "..\Timers\Timers.dll" no_namespace

int main(int argc, char* argv[])
{
  CoInitialize( NULL );
  try
  {
    IStopwatchPtr pStopwatch(__uuidof(Stopwatch));
    UseStopwatch(pStopwatch);
    pStopwatch=NULL;
  }
  catch(_com_error e)
  {
    std::cout<<"ERROR:"
       <<(LPCSTR)e.ErrorMessage()<<std::endl;
  }

  CoUninitialize();
  return 0;
}
```

Klient COM
Klient Stopwatch – inteligentny wskaźnik

```
void UseStopwatch(IStopwatchPtr pStopwatch)
{
  float nElapsedTime;
  float nOverhead;

  nOverhead = pStopwatch->Overhead;

  std::cout << "The overhead time without going across "
    "the COM boundary is " << nOverhead << std::endl;

  pStopwatch->Start();
  nElapsedTime = pStopwatch->ElapsedTime();

  std::cout << "The overhead time with the COM boundary is "
    << nElapsedTime << std::endl;

  std::cout << "The expense of using the COM boundary is "
    << (nElapsedTime - nOverhead) << std::endl;

  pStopwatch->Overhead = nElapsedTime;
}
```