DODATEK 3.A Język IDL

Język IDL został stworzony na potrzeby projektowania połączeń zdalnych wywołań procedur - RPC. Jest on wzorowany na języku C++, lecz nie jest bezpośrednio związany z żadnym językiem programowania. IDL nie służy także do tworzenia kodu źródłowego programów, tylko do projektowania zunifikowanych interfejsów, które to są odpowiednio tłumaczone na język, w którym powstaje aplikacja, przez specjalnie stworzone do tego celu translatory i generatory kodu. Stąd wynika pierwsza istotna własność tego języka: w IDL można projektować interfejsy dla dowolnych języków wyposażonych w odpowiedni translator.

Podjęta też została standaryzacja typów danych – pozwala to bowiem na uniknięcie problemów przy tworzeniu oprogramowania oraz podczas korzystania z niego przez różne maszyny / systemy. Przykładowo, nieporozumienia stwarzała odmienna kolejność bajtów młodszych i starszych w słowach różnych procesorów (np. firm Intel i Motorola), czy też różna długość typów danych, np. typ całkowity ma 2 albo 4 bajty długości. Z tego powodu językowi nadano drugą, bardzo ważną cechę: typy danych, występujące w IDL są ściśle określone.

3.1.1 Typy danych

W IDL występują sztywno ustalone, niezmienne pod względem rozmiaru typy danych. Jako norma kolejności bajtów w zmiennych wielobajtowych przyjęta została norma: Network Data Representation. Uzyskujemy przez to interfejs niezależny od systemu i maszyny. Zestaw zmiennych jest podany w tabeli 3.1:

Nazwa typu	***Rozmiar w bitach***	Uwagi
Boolean	TRUE lub FALSE	
Byte	8	Transmisja bez żadnych zmian kolejności bitów
Char	8	
Double	64	
Float	32	
Handle_t	32	Wskaźnik do przekazywanych danych poprzez RPC
Hyper	64	64 bitowa zmienna integer
Int	32	
Long	32	
Short	16	
Small	8	
Wchar_t	16	Znak w standardzie UNICODE (16 bit)

Tab. 3.1 – Typy danych w języku IDL

3.1.2 Wskaźniki i referencje

W języku IDL występują, podobnie jak w wielu językach, wskaźniki do zmiennych. Oznaczane są one przy pomocy znaku '*'. Wskaźniki, w zależności od użytej dyrektywy ***pointer_default( )*** mogą dzielić się pod względem właściwości na trzy typy (nazwy w zależności od parametru dyrektywy):

1. ref – wskaźniki używane jako referencje cechujące się:
   - zawsze wskazują na jakiś obiekt, czyli nie mogą być zerowe
   - nie mogą być aliasami
   - ich wskazanie nie zmienia się - wskazuje na to samo przed i po wywołaniu funkcji

2. unique – wskaźniki o następujących właściwościach:
   - mogą mieć wartość zerową
   - nie mogą być aliasami.
   - mogą zmieniać się w trakcie wywołania funkcji

3. ptr – wskaźniki podobne do wskaźników w języku C:
   - mogą mieć wartość zerową
   - mogą być aliasami
   - mogą się zmieniać w trakcie wywołania funkcji

W IDL są również przewidziane referencje zmiennych – oznacza się je, podobnie jak w C++, przy pomocy znaczka '&'.

3.1.3 Modyfikatory zmiennych

1) Modyfikatory kierunku przepływu danych:
W języku IDL argumenty funkcji interfejsu mogą mieć różne kierunki przepływu. Aby to sprecyzować, stosuje się następujące modyfikatory:

- **[in]** – argument jest podawany przez klienta komponentu
- **[out]** – argument jest zwracany przez komponent do wywołującego go klienta
- **[in, out]** – argument ma podwójne zastosowanie: jest zarówno podawany podczas wywołania funkcji przez klienta, jak i zwrócony przez serwer.

Przykład uzycia tych modyfikatorów jest podany poniżej:

```
[in ] long skladnik, [in] long skladnik, [out] long *suma, [in, out] byte *kod_operacji
```

Należy zwrócić uwagę na fakt, iż język IDL wymaga do przekazywania danych dla zmiennych typu **out** zadeklarowania ich jako wskaźnika do zmiennej. Powyższa własność została uwzględniona w podanym przykładzie.

2) Modyfikator **string**

Jedną z częściej występujących form przekazywania danych są łańcuchy danych, zwane stringami. Fizycznie stanowią pewną tablicę danych, której rozmiar jest wyznaczony pojawieniem się znaku o wartości zero na końcu – wyklucza to niestety możliwość użycia zera jako wartości w przekazywanych danych, lecz do zastosowań takich jak teksty jest to struktura wystarczająca. Wspieranie Unicodu, czyli dwubajtowej reprezentacji znaków, ma zastosowanie w opracowywaniu łańcuchów znaków w innych językach. Stringi zatem dotyczą zmiennych znakowych zarówno jedno- jak i dwubajtowych. Przykład użycia tego modyfikatora przedstawiony jest poniżej:

```
[ in, string ] wchar_t *str_wej, [ out, string ] char **str_wyj
```

3) Modyfikator **size_is()**

Często istnieje potrzeba przekazywania całych tablic danych przez interfejs. Problemem jest określenie wielkości przesyłanej tablicy – nie można użyć stringu, jeśli wartości tablicy mogą być zerowe. Aby ułatwić operację, wprowadzono modyfikator **size_is().** Użycie jego oznacza, że dana tablica ma rozmiar podawany przez inną zmienną, zamieszczoną w nawiasach modyfikatora. Przykład użycia takiej konstrukcji podany jest w podrozdziale 3.1.5.

3.1.4 Łączenie plików

Podobnie jak inne języki, w IDL można dołączać inne deklaracje, używając dyrektywy **import**, tak, jak to jest pokazane poniżej:

```
import  ''unknwn.idl'';
```

3.1.5 Tablice danych

Najpopularniejszą strukturą danych są tablice. Poniżej zostaną omówione sposoby przekazywania tablic danych w języku IDL:

1). Tablice o stałym rozmiarze

Standardowa metoda przekazywania tablic o stałym, ustalonym przez nas rozmiarze. Zasada ich definiowania jest podobna, jak w innych językach, co podkreśla przykład:

```
[ in ] long wektor[256], [ out ] long *wektorret
```

Pojawiła się tutaj wcześniej wspomniana metoda przekazywania parametru typu out:
jest on wskaźnikiem, w takiej formie jest też identyfikowany jako wskaźnik tablicy danych typu long.

2). Użycie dyrektywy size_is

Jest to rozwiązanie wymagające wprowadzenia dodatkowej zmiennej, której wartość będzie wskazywała na rozmiar

```
[ in ] long rozmiar, [ in, size_is( rozmiar ) ] long tablica[]
```

3). Stałe tablice o zmiennej liczbie (ważnych) elementów

Wykorzystuje się je w przypadkach, gdy liczba zwracanych danych może być mniejsza, niż podany rozmiar. Podaje się wtedy jako parametry stałą tablicę oraz dodatkową zmienną, wskazującą na liczbę ważnych elementów. Rozmiar tablicy jest dodatkowo opisana dyrektywą **length_is**. Zostało to zilustrowane przykładem:

```
[ out ] int * Wazne, [ out , length_is( *Wazne ) ] int Wektor[256]
```

W tym przykładzie tablica typu out nie była zdefiniowana jako wskaźnik, a mimo to była poprawna. Wynika to z faktu powiązań między tablicami a wskaźnikami w IDL, podobnymi do takowych, występujących w języku C/C++.

4). Tablice otwarte

W tej technice można, poprzez inne parametry wywołania, ustawiać rozmiar tablicy i rozkład danych w niej. Operacje dokonuje się przy pomocy dyrektyw:

- `size_is()` – podaje rozmiar tablicy
- `first_is()` – podaje pierwszy ważny element w tablicy
- `length_is()` – podaje długość ciągu ważnych danych

Tablica powinna być zadeklarowana wówczas jako wskaźnik. Oczywiście, należy zaznaczyć, że łańcuch danych, opisany przez dyrektywy first_is i length_is nie może wychodzić poza rozmiary tablicy. Tablice takie można definiować następująco:

```
[ in ] int rozmiar , [ in ] int pierwszy, [ in ] int dlugosc,
      [in, first_is(pierwszy), length_is(dlugosc), size_is(rozmiar)] int * tablica
```

3.1.6 Tablice wielowymiarowe

Tablice wielowymiarowe są także przewidziane przez standardy systemów rozproszonych jako struktury do przekazywania poprzez zdalne wywołania funkcji, zatem należy spodziewać się obecności ich specyfikacji w IDL. Jedynym problemem, jaki spotyka się przy definiowaniu tablic wielowymiarowych, jest interpretacja wskaźnika, czyli nazwy tablicy. Jest ona bowiem wielokrotnym wskazaniem, na końcu którego znajduje się dopiero element wskazywanego typu. Aby ustalić rozmiar danych, stosuje się wcześniej omówioną dyrektywę size_is() tylko w rozszerzonej postaci. Może ona bowiem przyjmować wektor parametrów, co opisuje liczbę możliwych (dostępnych) dla danej tablicy odwołań na kolejnych poziomach, czyli size_is(m, n) oznacza, że dana tablica ma m wskaźników do tablic n-wymiarowych. Przykładowo:

```
[ in , size_is(5,6) ] long ** tab
```

definiuje tablicę danych typu long o wymiarach 5 x 6, co jest fizycznie realizowane jako tablica pięciu wskaźników do tablic 6-cio elementowych, zwierających dane typu long.

3.1.7 Struktury danych

Podobnie jak w językach typu Pascal, Ada, Java czy C/C++, w IDL dane można organizować w struktury. Struktury te można później przekazywać poprzez interfejsy. Schemat zapisu struktury jest podobny do języka C, co pokazuje przykład:

```
typedef struct
{
      long x;
      long y;
}PunktXY;
```

użycie jej w funkcji deklaruje się następująco:

```
      Funkcja( [ in ] PunktXY pxy, [ out ] PunktXY * newpxy );
```

3.1.8 Projektowanie interfejsów

Głównym celem użycia języka IDL jest projektowanie interfejsów systemów rozproszonych. Wymaga to operowania na tablicach wskaźników do funkcji, zawartych w komponencie, wywoływanych przez klienta. Jednolitość takiej tablicy osiąga się poprzez połączenie funkcji w strukturę (istotna kolejność metod!). Poniżej zostanie podany i omówiony przykład definiowania takiego interfejsu:

```
[     object,
      uuid(0B01D6BF-6A7B-4695-B014-A21C9AB01EB7),
      helpstring("Interfejs przykladowy"),
      pointer_default(unique)
]
interface IPrzyklad : IUnknown
{
      import "unknwn.idl";
      HRESULT ZnajdzSlowo([in, string] char* slowo, [out, string] char  ** znaczenie) ;
};
```

Dane, znajdujące się między nawiasami kwadratowymi opisują charakterystyczne cechy interfejsu. Można (i to zazwyczaj się robi) podać w tym miejscu, że interfejs to pewien obiekt, ma swój (odpowiedni) unikalny globalnie numer identyfikacyjny **uuid**, pewną nazwę, skojarzoną z nim (nie musi być unikalna globalnie) oraz wskaźniki, jakie w nim występują, są typu unique. Następną sekcją danych jest struktura interfejsu. Interfejs IPrzyklad dziedziczy pewne własności po interfejsie IUnknown, którego definicja powinna być podana wcześniej (tu: w pliku ‘’unknwn.idl’’). W ciele interfejsu znajduje się tylko jedna funkcja – ZnajdzSlowo, mająca za parametry wywołania łańcuchy tekstowe wejściowy i wyjściowy.