Języki platformy .NET

Paweł Różański Wojciech Walewski

21 października 2005

Ostatnio chcieliśmy zdążyć opowiedzieć o:

- SML / NJ, SML.NET,
- OCaml, OCamlIL, F#,
- Nemerle,

a tym razem zdążyć opowiedzieć o:

- Nemerle cd.
- Python, IronPython,
- Boo,
- Lua,
- Ada, A#.

Ada

Ada

Ada

Krótkie wprowadzenie / przypomnienie:

- Składnia Pascalowa,
- *Case insensitive*,
- Komentarze: --,
- Tryby przekazywania parametrów: `in`, `out`, `in out`,

Ada

- Wyliczenia: `type State is {On, Off, Error};`
- Tablice: `type Tab is array(1..100) of Integer;`
- Wskaźniki: `type Tab_Ptr is access Tab;`
- Atrybuty: `State'First`, `State'Last`, `Tab'Length`,

```
for I in Tab'Range loop
    Tab(I) := 0;
end loop;
```

- Ograniczenia zakresu: `N : Integer range 0..1 := 0;`

Jak na razie to tylko posłodzony Pascal...

Ada

To co odróżnia Adę od Pascala, to:

- Zaprojektowana dla Departamentu Obrony USA, jako język dla systemów krytycznych,
- silne typowanie,
- rozbudowany system pakietów,
- wbudowane mechanizmy do programowania współbieżnego (zadaniowość),
- wyjątki.

Przykłady

A#

Rozszerzenia Ady dla .NET

A#

- Typy referencyjne, np. `MSSyst.Drawing.Image` mają zdefiniowane atrybuty `Image'Typ` i `Image'Ref`. Dla `MyImage:Image` trzeba stworzyć:

```
package MyImage
    type Typ;
    type Ref is access all Typ'Class;
    type Typ is new MSSyst.Object.Typ with record
        null;
    end record;
...
```

- Typy wartości mają zdefiniowany atrybut `ValueType` np. `MSSyst.Drawing.Point.ValueType`. Wtedy kompilator będzie generował dla nich kod jak dla typu wartości. Nie jest zdefiniowana referencja do tego typu.

A#

- Interfejsy

  ```
  type Typ(I_IContainerControl : IContainerControl.Ref) is new ...
  ```

  oznacza, że ten typ implementuje interfejs `IContainerControl`. Trzeba jeszcze zdefiniować typ `IContainerControl.Typ`:

  ```
  pragma MSIL_Interface(Typ);
  ```

- Konstruktory. Do ich określania służy `pragma`, np.

  ```
  pragma MSIL_Constructor(MyImage);
  ```

  Dodatkowo wprowadzono magiczną i nieintuicyjną składnię do tworzenia .NETowych klas pochodnych:

  ```
  function New_MyImage(This : Ref := null) return Ref is
      Super : Image.Ref := Image.New_Image(Image.Ref(This));
  begin
      return This;
  end New_MyImage;
  ```

A#

- Niejawna konwersja stringów. W .NET jest Unicode, w Adzie jest ASCII. Niejawna konwersja przeczy silnemu typowaniu, dlatego wprowadzono operator +.

```
Console.WriteLine( +"Hello world" );
```

  Jawne zadeklarowanie `use MSSyst.String`, powoduje niejawną konwersję z Ady do .NET. W drugą stronę konieczne jest explicité użycie +:

```
S : String := +Console.ReadLine;
```

A#

- Propercje w C# są cukrem syntaktycznym: dla propercji *P* istnieją: funkcja `get_`*P* i procedura `set_`*P*.
- Typy wyliczeniowe w .NET, w przeciwieństwie do Ady, mogą tworzyć wartości, które nie mają nazwy, np. wyliczenie `FontStyle` zawiera elementy **Bold** i *Italic*, ale można je dodać, tworząc ***Bold Italic***. Jak na razie, traktowane są jako stałe numeryczne i działa +.

A#

- Wywołania funkcji z windowsowych plików DLL:
  - Zwykle importy Ady nie generują kodu, ale importowanie funkcji bibliotecznej wymaga wygenerowania kodu.
  - Składnia w C#:

```
[DllImport("adagraph2001.dll")]
public static extern int CreateGraphWindow(int size);
```

  - W A#, w pliku specyfikacji:

```
function Open_Graph_Window(Size : in Integer) return Integer;
pragma Export(Stdcall, Open_Graph_Window,
    "[adagraph2001.dll]CreateGraphWindow");
```

  - W pliku źródłowym:

```
function Open_Graph_Window(Size : in Integer)
    return Integer is
begin
    return 0; -- ignorowane, ale kompilator sie cieszy
end Open_Graph_Window;
```

Przykłady