helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

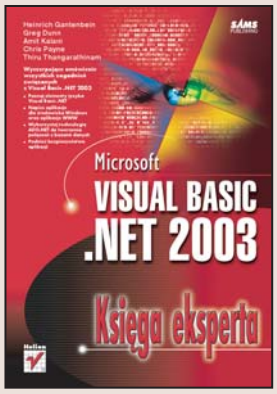
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

Microsoft Visual Basic .NET 2003. Księga eksperta



Autorzy: Heinrich Gantenbein, Greg Dunn,
Amit Kalani, Chris Payne, Thiru Thangarathinam
Tłumaczenie: Paweł Gonera (wstęp, rozdz. 1–22),
Tomasz Walczak (rozdz. 23–40)
ISBN: 83-246-0096-5
Tytuł oryginału: Microsoft Visual Basic .NET 2003 Unleashed
Format: B5, stron: 976

Wyczerpujące omówienie wszystkich zagadnień związanych z Visual Basic .NET 2003

- Poznaj elementy języka Visual Basic .NET
- Napisz aplikacje dla środowiska Windows oraz aplikacje WWW
- Wykorzystaj technologię ADO.NET do tworzenia połączeń z bazami danych
- Podnieś bezpieczeństwo aplikacji

Visual Basic to jeden z najpopularniejszych języków programowania. Jego pojawienie się na rynku zrewolucjonizowało proces tworzenia aplikacji dla Windows – model programowania był tak prosty, że pozwalał nawet niezbyt wprawnym programistom na tworzenie rozbudowanych aplikacji. Kolejne wersje tego języka programowania posiadały coraz większe możliwości. Ukoronowaniem rozwoju Visual Basica było umieszczenie go w środowisku .NET jako jednego z dostępnych w nim języków programowania. Dzięki integracji z .NET Visual Basic stał się w pełni obiektowym językiem, pozwalającym na zrealizowanie zarówno prostych aplikacji Windows i WWW, jak i złożonych wielowątkowych systemów rozproszonych.

Książka „Microsoft Visual Basic .NET 2003. Księga eksperta" przedstawia wszystko, co jest związane z językiem Visual Basic 2003 oraz tworzeniem za jego pomocą aplikacji w środowisku .NET. Opisuje podstawy języka, zasady programowania w nim oraz techniki obiektowe. Czytając ją, dowiesz się, jak tworzyć aplikacje dla systemu Windows oraz aplikacje WWW, łączyć aplikacje z bazami danych, zabezpieczać je i zwiększać ich wydajność. Nauczysz się stosować usługi sieciowe do przesyłania danych przez sieć i łączenia aplikacji z innymi. Poznasz wszystkie nowoczesne metody programowania.

- Składniki platformy .NET
- Podstawy języka Visual Basic i programowania obiektowego
- Tworzenie aplikacji Windows Forms
- Korzystanie z funkcji graficznych
- Budowanie wersji instalacyjnych
- Połączenia z bazami danych za pomocą ADO.NET oraz języka XML
- Tworzenie aplikacji WWW z wykorzystaniem ASP.NET
- Operacje wejścia i wyjścia
- Programowanie wielowątkowe
- Zabezpieczanie aplikacji
- Usługi sieciowe

Wykorzystaj ogromne możliwości środowiska .NET
i stwórz wydajne, stabilne i bezpieczne aplikacje

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

O autorach 21

Wstęp 23

Historia języka Visual Basic (VB) 23
Visual Basic .NET (oraz C#) 24
Visual Basic .NET 2003. Księga eksperta 24
Sposób organizacji książki 24
Część I „Podstawy” 24
Część II „Aplikacje Windows Forms” 25
Część III „Programowanie baz danych” 25
Część IV „Aplikacje Web (ASP.NET)” 25
Część V „Zaawansowane programowanie” 25
Część VI „Zabezpieczanie aplikacji” 25
Część VII „Technologia Remoting” 26
Część VIII „Usługi Web” 26
Część IX „Usługi Enterprise” 26
Jak korzystać z tej książki 26

Część I Podstawy 27

Rozdział 1. Wprowadzenie do .NET 29

Oprogramowanie jako usługa 29
Produkty .NET 30
Usługi .NET 30
.NET Framework 30
Zadania .NET Framework 33
Poznajemy architekturę .NET Framework 34
Poznajemy zadania .NET Framework 35
Co to jest MSIL? 36
Co to jest plik Portable Executable (PE)? 36
Poznajemy kompilator JIT 37
Wspólne środowisko uruchomieniowe 37
Składniki CLR 37
Biblioteka klas .NET Framework 38
Zalety CLR 41
Co to jest podzespół? 43
Struktura podzespołu 44
Wykorzystanie metody XCOPY do bezobsługowej instalacji podzespołów 45

Problemy przy projektowaniu podzespołu ... 46
Typy podzespołów ... 46
Metadane ... 48
System wspólnego języka (CLS) ... 49
Podsumowanie ... 50
Propozycje dalszych lektur ... 50

Rozdział 2. Podstawy języka ... 51

Deklarowanie zmiennych ... 51
Znaczenie słów kluczowych ... 52
Poziomy deklaracji ... 52
Deklaracja typu danych ... 53
Deklaracja czasu życia ... 53
Deklaracja zasięgu ... 53
Deklaracja dostępności ... 54
Zmienne i przypisywanie ... 54
Użycie przestrzeni nazw ... 55
Słowo kluczowe Imports ... 56
Zastosowanie synonimów ... 58
Tworzenie modułów w Visual Basic .NET ... 58
Tworzenie wielu modułów ... 59
Struktury sterujące ... 60
Struktura wyboru If...Then ... 61
Struktura wyboru If...Then...Else ... 61
Instrukcja Select...Case ... 61
Pętla While ... 62
Pętla DoWhile...Loop ... 63
Pętla Do...LoopWhile ... 63
Pętla DoUntil...Loop ... 64
Pętla Do...LoopUntil ... 64
Pętla For...Next ... 64
Pętla ForEach...Next ... 65
Instrukcja Exit ... 66
Operatory przypisania ... 67
Operatory logiczne ... 67
Instrukcje warunkowe ... 70
Tworzenie wyrażeń warunkowych i ich zastosowanie w instrukcjach warunkowych ... 70
Użycie operatora przypisania ... 71
Stałe ... 71
Użycie operatorów ... 72
Funkcje wbudowane ... 73
Funkcje konwersji ... 73
Funkcje operujące na ciągach znaków ... 74
Inne przydatne funkcje ... 74
Tworzenie własnych podprogramów ... 74
Procedury ... 76
Funkcje ... 76
Zasięg zmiennych ... 77
Dlaczego zasięg jest tak ważny? ... 77
Podsumowanie ... 78
Propozycje dalszych lektur ... 78

Rozdział 3. Podstawy programowania obiektowego 79
Mechanizmy obiektowe 79
Tworzenie klas 80
Tworzenie prostych klas 80
Obiekty, klasy i instancje 82
Tworzenie konstruktorów 83
Składniki obiektu 84
Operacje na obiektach 88
Deklaracja i tworzenie obiektów 88
Referencje do obiektów 89
Dereferencja obiektów 90
Obsługa zdarzeń 90
Obsługa pojedynczego zdarzenia 91
Obsługa wielu zdarzeń 91
Słowo kluczowe WithEvents 92
Generowanie zdarzeń 92
Odbieranie zdarzeń 93
Dziedziczenie w Visual Basic .NET 96
Przesłanianie metod 97
Podsumowanie 98
Propozycje dalszych lektur 99

Rozdział 4. Typy danych w .NET Framework 101
Co to jest wspólny system typów? 102
System.Object — od tego się zaczęło 102
Typy i synonimy 103
Poznajemy typy danych 104
Odzyskiwanie nieużytków 105
Typy wartościowe 106
Typy referencyjne 112
Klasa System.String 113
Modyfikowalne ciągi znaków 114
Obiekt StringBuilder 114
Co to jest GUID? 119
Pakowanie i rozpakowywanie 120
Podsumowanie 120
Propozycje dalszych lektur 121

Część II Aplikacje Windows Forms 123

Rozdział 5. Windows Forms 125
Tworzenie nowego projektu 125
Sposób uruchomienia projektu 126
Wprowadzenie do formularzy 127
Klasa Form 127
Zmiana właściwości formularza 130
Zmiana nazwy formularza 131
Wyświetlanie tekstu w pasku tytułowym formularza 131
Zmiana koloru tła formularza 132

Umieszczanie rysunku jako tła formularza ... 133
Wyświetlanie ikony w formularzu ... 133
Określanie początkowego położenia formularza ... 134
Określanie rozmiaru oraz położenia formularza ... 134
Obiekt Application ... 136
Sterowanie wyglądem formularza ... 136
Pokazywanie i usuwanie formularzy ... 138
Pokazywanie formularzy ... 138
Konfiguracja przycisków sterujących ... 142
Metody formularza ... 144
Obsługa problemów z wyświetlaniem ... 144
Ponowne inicjowanie właściwości ... 145
Obsługa zdarzeń ... 145
Pętla komunikatów ... 146
Zdarzenia formularza ... 147
Sterowanie działaniem programu ... 147
Zdarzenia klawiatury i myszy ... 149
Podsumowanie ... 151
Propozycje dalszych lektur ... 152

Rozdział 6. Formanty ... 153

Wykorzystanie narzędzia Form Designer w Visual Studio .NET ... 153
Okno Toolbox ... 154
Okno Properties ... 154
Dodawanie procedur obsługi zdarzeń ... 155
Operacje na formantach formularza ... 156
Formant TextBox ... 157
Tworzenie pól do wprowadzania hasła ... 157
Tworzenie pola tekstowego tylko do odczytu ... 158
Zdarzenia generowane przez formant TextBox ... 158
Formant Button ... 159
Użycie formantu Button ... 160
Obsługa kliknięcia przycisku ... 160
Formant Label ... 161
Formant RadioButton ... 162
Formant CheckBox ... 163
Reagowanie na kliknięcie pola wyboru ... 164
Formant ListBox ... 165
Wybieranie elementów formantu ListBox ... 165
Wyszukiwanie elementów w formancie ListBox ... 165
Manipulowanie elementami w formancie ListBox ... 166
Wspólne metody dostępne w formantach list ... 169
Formant ComboBox ... 169
Menu ... 173
Budowanie grup menu ... 173
Dodawanie poziomów menu ... 174
Programowe tworzenie menu ... 175
Dodawanie skrótów klawiszowych ... 177
Dodawanie klawiszy szybkiego dostępu ... 178
Obsługa zdarzenia Click ... 179

Zdarzenia sterujące interfejsu użytkownika dla obiektów MenuItems ... 180
Definiowanie obiektu MenuItem jako separatora ... 180
Układ elementów menu ... 181
Menu rozmieszczane od prawej do lewej ... 182
Menu kontekstowe ... 182
Podsumowanie ... 183
Propozycje dalszych lektur ... 184

Rozdział 7. Rysowanie ... 185

Interfejs urządzeń graficznych (GDI) ... 185
GDI+: API wyższego poziomu ... 186
Własności GDI+ ... 186
Przestrzenie GDI+ w .NET ... 186
Przestrzeń nazw System.Drawing ... 187
Przestrzeń nazw System.Drawing.Design ... 187
Przestrzeń nazw System.Drawing.Printing ... 188
Przestrzeń nazw System.Drawing.Imaging ... 188
Przestrzeń nazw System.Drawing.Drawing2D ... 189
Przestrzeń nazw System.Drawing.Text ... 189
Klasa Graphics ... 189
Często używane obiekty graficzne ... 190
Rysowanie linii ... 192
Rysowanie prostych figur ... 195
Wypełnianie figur ... 198
Zbiory figur ... 200
Operacje na obrazach ... 204
Jednostki miary ... 205
Operacje na czcionkach ... 205
Podsumowanie ... 206
Propozycje dalszych lektur ... 206

Rozdział 8. Instalowanie aplikacji ... 207

Wprowadzenie do instalacji ... 207
Planowanie instalacji ... 208
Struktura podzespołu ... 208
Opcje instalacji obsługiwane przez .NET ... 209
Zastosowanie XCOPY do instalowania aplikacji ... 209
Zastosowanie programu Visual Studio .NET Installer ... 209
Różnice między XCOPY i instalatorem Windows ... 211
Typy szablonów projektów instalacyjnych dostępnych w Visual Studio .NET ... 212
Tworzenie pakietu instalacyjnego ... 213
Konfigurowanie właściwości instalacji ... 214
Konfigurowanie właściwości projektu ... 215
Typy edytorów instalacji ... 217
Konfiguracja edytora systemu plików ... 217
Konfigurowanie edytora rejestru ... 220
Konfiguracja edytora typów plików ... 220
Konfigurowanie edytora własnych akcji ... 221
Konfigurowanie edytora warunków uruchomienia ... 222
Konfigurowanie edytora interfejsu użytkownika ... 223

Generowanie pakietu instalacyjnego ... 226
Instalowanie aplikacji ... 226
Odinstalowanie aplikacji ... 227
Podsumowanie ... 228
Propozycje dalszych lektur ... 228

Rozdział 9. Drukowanie ... 229
Drukowanie w .NET ... 229
Definiowanie dokumentu ... 231
Zdarzenia obsługiwane przez klasę PrintDocument ... 232
Metody klasy PrintDocument ... 233
Drukowanie dokumentu ... 233
Zastosowanie okien dialogowych drukowania ... 235
Zastosowanie formantu PrintPreviewDialog ... 236
Zastosowanie formantu PrintDialog ... 238
Użycie formantu PageSetupDialog ... 240
Zastosowanie formantu PrintPreviewControl ... 243
Obsługa ustawień ... 244
Klasa PrinterSettings ... 245
Rola klasy PageSettings w drukowaniu ... 246
Podsumowanie ... 247
Propozycje dalszych lektur ... 247

Rozdział 10. Zaawansowane techniki Windows Forms ... 249
Dynamiczne dodawanie formantów ... 249
Dynamiczne przechwytywanie zdarzeń ... 251
Tworzenie formularzy zależnych ... 252
Interfejs MDI ... 253
Implementacja dziedziczenia w Visual Studio .NET ... 254
Formant NotifyIcon ... 255
Zastosowanie dostawców w aplikacjach Windows Forms ... 257
Formant HelpProvider ... 257
Dodawanie pomocy do aplikacji Windows Forms ... 257
Wyświetlanie pomocy z użyciem klasy ToolTips ... 262
Okna dialogowe ... 264
Użycie kursorów w aplikacjach Windows Forms ... 265
Zaawansowane formanty interfejsu użytkownika ... 267
Formant TreeView ... 267
Formant DateTimePicker ... 272
Formant MonthCalendar ... 273
Podsumowanie ... 275
Propozycje dalszych lektur ... 275

Część III Programowanie baz danych ... 277

Rozdział 11. Klasy ADO.NET ... 279
Dostawcy danych ... 279
Dostawcy SqlClient oraz OleDb ... 280
Obiekt Connection ... 281
Właściwość ConnectionString ... 281
Metoda Open() ... 282

Metoda Close() ... 283
Metoda BeginTransaction() ... 283
Obiekt Command ... 289
Przygotowanie obiektu Command do wykonania instrukcji SQL ... 289
Przygotowanie obiektu Command do wywołania procedury składowanej ... 291
Przygotowanie obiektu Command do wywołania TableDirect ... 293
Metody Execute obiektu Command ... 293
Obiekt DataReader ... 294
Obiekt DataSet ... 297
Obiekty DataSet ze ścisłą i luźną kontrolą typów ... 298
Obiekty podrzędne do DataSet ... 300
Obiekt DataView ... 311
Obiekt DataAdapter ... 312
Użycie kreatora DataAdapter do generowania ciągu połączenia ... 313
Problemy z wielodostępem ... 314
Przykładowe aplikacje dołączone do książki ... 315
Aplikacja TransactionsAndDataSets ... 315
Aplikacja TransactionInteractionExplorer ... 315
Aplikacja ConcurrencyViolator ... 318
Podsumowanie ... 319
Propozycje dalszych lektur ... 320

Rozdział 12. Tworzenie i obsługa baz danych w Visual Studio ... 321

Tworzenie bazy danych ... 322
Tworzenie i modyfikacja komponentów bazy danych ... 323
Tworzenie nowej tabeli ... 323
Projekt tabeli Instructor ... 325
Definiowanie klucza głównego ... 326
Rodzaje kluczy głównych ... 327
Pełny projekt tabeli Instructor ... 330
Zapisywanie tabeli Instructor ... 332
Projekt tabeli Student ... 333
Projekt tabeli Course ... 333
Projekt tabeli Enrollment ... 334
Modyfikacja projektu tabeli ... 334
Kilka słów o typach danych dla użytkowników Microsoft Access ... 337
Pobieranie danych z tabeli ... 338
Diagram bazy danych School ... 338
Tworzenie nowej relacji ... 341
Relacja Course-Instructor ... 343
Relacja Enrollment-Course ... 345
Relacja Enrollment-Student ... 345
Pełny diagram bazy danych ... 346
Zapisywanie diagramu bazy danych ... 346
Przeglądanie właściwości tabel ... 347
Tworzenie nowej perspektywy ... 349
Ukrywanie paneli w oknie projektowania perspektywy ... 351
Wyświetlanie rekordów z perspektywy ... 355
Zapisywanie perspektywy ... 355
Odczyt danych z zamkniętej perspektywy ... 357
Uwagi dotyczące uporządkowania perspektyw ... 358

Tworzenie procedur składowanych ... 360
Edycja procedur składowanych ... 362
Tworzenie procedury składowanej z użyciem SqlDataAdapter ... 364
Tworzenie funkcji ... 364
Operacje na bazie danych, jakich nie da się wykonać w Visual Studio ... 368
Podsumowanie ... 369
Propozycje dalszych lektur ... 369

Rozdział 13. Użycie kreatorów Data Form oraz DataAdapter w aplikacjach Windows ... 371

Opis kreatora ... 372
Tworzenie formularza danych ... 373
Obiekty generowane przez kreator ... 377
Analiza wygenerowanego kodu ... 380
Deklaracje zmiennych ... 380
Tworzenie i inicjalizacja ... 381
Metody tworzone przez kreator Data Form ... 387
Metody wykorzystywane do obsługi zdarzeń ... 387
Metody niewykorzystywane do obsługi zdarzeń ... 389
Przegląd kodu realizującego różne funkcje ... 390
Ładowanie danych ... 390
Modyfikacja danych ... 393
Anulowanie wszystkich zmian ... 395
Dodawanie rekordów do tabeli Publishers ... 395
Kasowanie rekordu z tabeli Publishers ... 395
Anulowanie zmian w rekordzie tabeli Publishers ... 396
Nawigacja w tabeli Publishers ... 396
Uwalniamy się od kreatora Data Form ... 397
Tworzenie formularza danych SqlClient z wykorzystaniem kreatorów Data Form oraz SqlDataAdapter ... 397
Następne kroki ... 405
Podsumowanie ... 406
Propozycje dalszych lektur ... 406

Rozdział 14. Dołączanie danych w aplikacjach Windows ... 407

Przegląd dołączania danych ... 408
Kategorie łączenia danych: tylko odczyt kontra odczyt-zapis oraz proste kontra złożone ... 409
Potencjalne źródła danych oraz obsługa interfejsu IList ... 410
Klasy .NET implementujące interfejs IList ... 410
BindingContext ... 410
Obiekt CurrencyManager ... 412
Obiekt PropertyManager ... 413
Obiekty dołączane do danych ... 414
Proste dołączanie — formant TextBox ... 415
Złożone dołączanie — formant ComboBox ... 415
Formant DataGrid ... 416
Dopasowywanie źródła danych w instrukcjach nawigacyjnych do źródła danych dołączenia ... 417
Dołączanie formantów do obiektu DataSet ... 418
Dołączanie właściwości obiektu do wartości tabelowych ... 420
Ważne metody klas BindingContext i DataSet ... 423
Zastosowanie obiektu CurrencyManager do łączenia nietabelowego ... 427

Dołączanie wielu formantów formularza do różnych właściwości jednego obiektu ... 428
Dołączanie wielu właściwości jednego obiektu do wielu właściwości innego obiektu ... 430
Dołączanie formantów formularza do listy ArrayList obiektów ... 431
Ograniczenie aplikacji BindToAnArraylistOfObjects tylko do odczytu ... 433
Dołączanie do odczytu i zapisu ... 433
Implementacja dołączania do odczytu i zapisu we własnej kolekcji obiektów ... 434
Przykładowa aplikacja: ReadWriteBindableCollection ... 435
Jak działa aplikacja ReadWriteBindableCollection ... 436
Klasa Product ... 439
Klasa Products ... 443
Dodatkowe operacje pozwalające podłączyć aplikację ReadWriteBindableCollection do bazy danych ... 450
Przykładowe aplikacje dołączone do książki ... 451
Podsumowanie ... 451
Propozycje dalszych lektur ... 453

Rozdział 15. ADO.NET i XML ... 455
Wstęp ... 455
Metody związane z XML w klasie DataSet ... 457
Odczyt i zapis schematu XML ... 457
Wpływ danych schematu na dane wyświetlane w DataGrid ... 459
Odczyt danych XML ... 460
Aplikacja ReadXmlWorkoutMachine ... 461
Zapis danych XML ... 463
Przykładowe wyniki działania metody WriteXml() ... 464
Tworzenie obiektu XmlReader z wykorzystaniem obiektu Command ... 467
Zapis danych schematu do obiektu DataSet za pomocą DataAdapter ... 469
Przykładowe aplikacje dołączone do książki ... 471
Pozostałe mechanizmy wspierające XML w .NET Framework i Visual Studio ... 472
Podsumowanie ... 472
Propozycje dalszych lektur ... 472

Część IV Aplikacje Web (ASP.NET) ... 475

Rozdział 16. Proste aplikacje Web ... 477
Tworzenie prostej aplikacji Web ... 477
Przygotowywanie serwera WWW ... 478
Konfigurowanie bazy danych ... 479
Tworzenie strony domowej blogu ... 480
Tworzenie formularzy Web ... 486
Obsługa danych w sieci WWW ... 486
Użycie formularzy ukrytego kodu ... 487
Tworzenie interakcyjnych formularzy Web ... 489
Zastosowanie formantów Web na formularzach ... 495
Poznajemy formanty Web ... 495
Tworzenie rozbudowanego interfejsu użytkownika ... 498
Obsługa zdarzeń Web ... 502
Zdarzenia ASP.NET — powtórka ... 503
Poznajemy obsługę zdarzeń ... 505
Podsumowanie ... 511
Propozycje dalszych lektur ... 511

Rozdział 17. Dołączanie danych w aplikacjach Web 513
Dołączanie danych do prostych formantów 513
Zastosowanie formantów szablonowych do wyświetlania danych 517
Propagacja zdarzeń 518
Formant Repeater 519
Formant DataList 522
Formant DataGrid 526
Tworzenie dołączalnych formantów Web 530
Przetwarzanie przesyłanych danych formularza 534
Model przetwarzania danych formularza 534
Modyfikujemy formant LoginForm 536
Podsumowanie 539
Propozycje dalszych lektur 539

Rozdział 18. Zarządzanie stanem 541
Zapamiętywanie danych na poziomie aplikacji 542
Zapamiętywanie danych sesji 544
Przypomnienie zagadnienia sesji Web 544
Obsługa sesji 545
Zapamiętywanie stanu widoku 546
Zarządzanie mechanizmem cookie 551
Przypomnienie na temat cookie 551
Operacje na cookie 552
Podsumowanie 556
Propozycje dalszych lektur 556

Rozdział 19. Buforowanie 557
Buforowanie elementów statycznych 557
Buforowanie całych stron 561
Buforowanie formantów 562
Buforowanie deklaratywne 562
Buforowanie programowe 564
Podsumowanie 565
Propozycje dalszych lektur 565

Rozdział 20. Zarządzanie witryną WWW 567
Konfigurowanie witryny 567
Plik global.asax 568
Plik web.config 570
Zastosowanie pamięci podręcznej podzespołów dla plików aplikacji 574
Śledzenie aplikacji ASP.NET 575
Podsumowanie 578
Propozycje dalszych lektur 578

Rozdział 21. Bezpieczeństwo 579
Przegląd zabezpieczeń w ASP.NET 579
Uwierzytelnianie użytkownika Web 581
Uwierzytelnianie Windows 582
Uwierzytelnianie typu Forms 584
Uwierzytelnianie za pomocą usługi Passport 587

Autoryzacja użytkowników i ról ... 588
Autoryzacja bazująca na rolach ... 588
Autoryzacja bazująca na zasobach ... 592
Zapobieganie atakom wstrzyknięcia kodu ... 593
Zabezpieczanie ciągów połączenia i innych kluczy ... 595
Podsumowanie ... 596
Propozycje dalszych lektur ... 597

Rozdział 22. Rozszerzanie ASP.NET ... 599
Poznajemy sposób obsługi strumienia HTTP w ASP.NET ... 599
Zastosowanie modułów HTTP do modyfikacji danych wyjściowych ... 602
Tworzenie własnych obiektów HttpHandler ... 605
Podsumowanie ... 608
Propozycje dalszych lektur ... 608

Część V Zaawansowane programowanie ... 609

Rozdział 23. Wersjonowanie i globalna pamięć podręczna podzespołów ... 611
Wprowadzenie do wersjonowania platformy .NET ... 612
Podzespoły prywatne ... 612
Podzespoły współdzielone ... 613
Przypisywanie silnych nazw ... 613
Globalna podręczna pamięć podzespołów ... 613
Używanie podzespołów współdzielonych ... 614
Przypisywanie silnej nazwy do podzespołu ... 614
Dodawanie podzespołu do GAC ... 616
Opóźnione podpisywanie podzespołów ... 618
Strategie dołączania podzespołów ... 620
Sprawdzanie strategii z poziomu aplikacji ... 621
Sprawdzanie strategii z poziomu wydawcy ... 623
Sprawdzanie strategii z poziomu administratora ... 624
Jak wspólne środowisko uruchomieniowe dołącza podzespół ... 625
Podsumowanie ... 627
Propozycje dalszych lektur ... 627

Rozdział 24. Operacje wejścia-wyjścia oraz utrwalanie obiektów ... 629
Operacje wejścia-wyjścia na plikach ... 629
Używanie klasy File ... 630
Używanie klasy FileStream ... 630
Używanie klas StreamReader oraz StreamWriter ... 631
Używanie klas BinaryReader oraz BinaryWriter ... 633
Serializacja ... 634
Serializacja XML ... 636
Serializacja w czasie wykonywania programu ... 639
Podsumowanie ... 645
Propozycje dalszych lektur ... 646

Rozdział 25. Współpraca z modelem COM 647
Wprowadzenie do współdziałania .NET Framework z modelem COM 647
Jak .NET Framework współdziała z modelem COM? 648
Używanie komponentów COM w aplikacjach .NET Framework 649
Tworzenie i rejestrowanie komponentów COM 649
Tworzenie RCW dla komponentu COM 650
Automatyczne tworzenie opakowania RCW 651
Używanie komponentów .NET Framework w aplikacjach COM 652
Tworzenie za pomocą Visual Basic .NET klasy dostępnej dla aplikacji COM 652
Tworzenie opakowania wywoływanego z COM (CCW) 654
Używanie komponentów .NET Framework w aplikacji COM 655
Podsumowanie 657
Propozycje dalszych lektur 657

Rozdział 26. Zdarzenia i delegacje 659
Wprowadzenie do zdarzeń 659
Definiowanie i zgłaszanie zdarzeń 660
Obsługa zdarzeń 661
Wprowadzenie do delegacji 663
Definiowanie delegacji 664
Tworzenie instancji delegacji 665
Wywoływanie delegacji 665
Delegacje zbiorowe 666
Zdarzenia i delegacje 667
Podsumowanie 668
Propozycje dalszych lektur 668

Rozdział 27. Wielowątkowość 669
Wprowadzenie do wątków 669
Zalety wielowątkowości 670
Wady wielowątkowości 670
Używanie wątków 671
Tworzenie i uruchamianie wątków 671
Właściwości wątków 672
Metody wątków 673
Synchronizacja wątków 674
Sytuacja wyścigu 674
Zakleszczenie 674
Sposoby synchronizacji 675
Programowanie asynchroniczne 678
Model programowania asynchronicznego .NET Framework 678
Pobieranie wyników wywołań metod asynchronicznych 679
Asynchroniczne wywołania zwrotne 679
Podsumowanie 680
Propozycje dalszych lektur 681

Rozdział 28. Rozszerzanie możliwości platformy za pomocą refleksji 683
Metadane 683
Wprowadzenie do mechanizmu refleksji 685
Stosowanie refleksji do podzespołu 685

Stosowanie refleksji do typu ... 686
Późne dołączanie ... 689
Podsumowanie ... 690
Propozycje dalszych lektur ... 690

Część VI Zabezpieczanie aplikacji ... 691

Rozdział 29. Bezpieczeństwo oparte na uprawnieniach kodu (CAS) ... 693

Zarządzanie uprawnieniami kodu ... 694
Sprawdzanie domyślnych zabezpieczeń za pomocą narzędzia konfiguracyjnego .NET Framework ... 694
Maksymalizacja bezpieczeństwa systemu ... 696
Łagodzenie systemu bezpieczeństwa w celu tworzenia oprogramowania ... 704
Instalacja zasad CAS ... 704
Jak środowisko CLR określa zbiór uprawnień nadawanych podzespołowi? ... 705
Określanie uprawnień nadanych podzespołowi przez zasady ... 705
Stosowanie atrybutów z poziomu podzespołu do zmiany nadawanych uprawnień ... 707
Jak używać żądań zabezpieczeń? ... 708
Proste żądania i analiza stosu ... 709
Żądania z poziomu dołączania ... 712
Żądania z poziomu dziedziczenia ... 713
Podpisywanie podzespołów ... 714
Natychmiastowe podpisywanie ... 715
Opóźnione podpisywanie ... 715
Łączenie różnych technik ... 717
Podsumowanie ... 717
Propozycje dalszych lektur ... 717

Rozdział 30. Uprawnienia oparte na tożsamości i rolach ... 719

Autoryzacja i uwierzytelnianie ... 719
Uwierzytelnieni użytkownicy ... 720
Bezpieczeństwo oparte na rolach ... 721
Deklaratywne zabezpieczanie oparte na rolach ... 722
Programowe zabezpieczanie oparte na rolach ... 723
Tworzenie własnych obiektów kontekstu bezpieczeństwa i tożsamości ... 724
Podsumowanie ... 729
Propozycje dalszych lektur ... 729

Część VII Technologia Remoting ... 731

Rozdział 31. Technologia Remoting w praktyce ... 733

Wprowadzenie do technologii Remoting ... 733
Procesy systemu Windows a domeny aplikacji .NET Framework ... 734
Zdalne korzystanie z obiektów i szeregowanie ... 734
Architektura technologii Remoting ... 736
Proste zdalne korzystanie z obiektów w obrębie procesu ... 737
Jawne zdalne korzystanie z obiektów ... 740
Typy aktywacji zdalnych obiektów ... 740
Zdalne korzystanie z obiektów konfigurowane programowo ... 741
Deklaratywne konfigurowanie ... 753

Bezpieczeństwo zdalnych wywołań ... 760
Ograniczenia parametrów zdalnych wywołań ... 760
Zarządzanie cyklem życia zdalnych obiektów ... 762
Jak zdalne obiekty kontrolują własną dzierżawę? ... 764
Jawne odnawianie dzierżawy ... 764
Odnawianie dzierżawy za pomocą sponsora ... 765
Zalecane techniki kontroli cyklu życia ... 771
Jednostronne wywołania i zdalne zdarzenia ... 771
Konteksty i usługi ... 772
Klasy Object, MarshalByRefObject i ContextBoundObject ... 773
Usługi ... 773
Podsumowanie ... 773
Propozycje dalszych lektur ... 774

Część VIII Usługi Web ... 775

Rozdział 32. Podstawowe usługi Web ... 777
Protokoły wykorzystywane przez usługi Web ... 778
Tworzenie usług Web ... 778
Testowanie usługi ... 779
Wywoływanie usług Web ... 782
Tworzenie usług Web używających złożonych parametrów ... 786
Importowanie usług Web ... 789
Wykrywanie usług ... 789
Tworzenie pośrednika usługi ... 789
Dostosowywanie usług Web do własnych potrzeb ... 790
Atrybut WebService ... 790
Atrybut WebMethod ... 790
Tworzenie własnych nagłówków SOAP ... 793
Podsumowanie ... 796
Propozycje dalszych lektur ... 796

Rozdział 33. Bezpieczeństwo i rozszerzenia usług Web ... 797
Zabezpieczenia z poziomu platformy ... 797
Strategie uwierzytelniania IIS ... 798
Strategie uwierzytelniania ASP.NET ... 798
Przekazywanie informacji o uwierzytelnieniu do usług Web ... 800
Autoryzacja usług Web ... 800
Zabezpieczenia z poziomu komunikatu ... 802
Dodawanie obsługi WSE do aplikacji ... 802
Uwierzytelnianie za pomocą WSE ... 803
Żetony zabezpieczeń WSE ... 806
Autoryzacja za pomocą WSE ... 806
Zasady bezpieczeństwa WSE ... 806
Podsumowanie ... 808
Propozycje dalszych lektur ... 809

Część IX Usługi Enterprise 811

Rozdział 34. Wprowadzenie do usług Enterprise 813

Ulepszanie oprogramowania za pomocą architektur n-warstwowych 813
Definiowanie komponentów za pomocą interfejsów 814
Wprowadzenie do architektury usług Enterprise 815
Tworzenie prostych obiektów serwerowych 816
Rozszerzenie usług składowych w konsoli MMC 822
Debugowanie obiektów serwerowych 824
Instalacja zdalna i lokalna 825
Usprawnianie pracy z COM+ 826
Dodawanie instrumentacji i dokumentacji do obiektów serwerowych 833
Podsumowanie 833
Propozycje dalszych lektur 833

Rozdział 35. Usługi związane z aktywacją 835

Zwiększanie skalowalności za pomocą aktywacji w czasie trwania (JIT) 835
Implementacja interfejsu COM+ IObjectControl 839
Poprawa wydajności za pomocą tworzenia puli obiektów 841
Łączenie aktywacji JITA z tworzeniem puli obiektów w celu poprawy skalowalności i wydajności 847
Zmiana wartości parametrów ObjectPooling w czasie instalacji 848
Przekazywanie argumentów czasu wykonania za pomocą ciągu znaków konstrukcji 849
Włączanie równoważenia obciążenia 852
Podsumowanie 852
Propozycje dalszych lektur 852

Rozdział 36. Usługi związane z transakcjami 853

Upraszczanie kodu transakcyjnego za pomocą transakcji COM+ 853
Transakcje i poziom izolacji 855
Implementacja komponentu transakcyjnego 857
Kontrola granic transakcji i poziom izolacji 862
Rozluźnianie wymogu izolacji za pomocą właściwości IsolationLevel 862
Kontrola granic transakcji 863
Łączenie transakcji z pulą obiektów i ciągami znaków konstrukcji 864
Używanie stron internetowych i usług Web jako głównych obiektów transakcji 867
Podsumowanie 867
Propozycje dalszych lektur 867

Rozdział 37. Zabezpieczanie przed jednoczesnym dostępem 869

Wielowątkowość i zabezpieczanie zasobów 869
Poprawianie rozwiązania za pomocą blokad wątków logicznych opartych na czynności 870
Kontrolowanie granic czynności 871
Łączenie czynności z aktywacją JIT, transakcjami i korzystaniem z puli 872
Podsumowanie 873
Propozycje dalszych lektur 873

Rozdział 38. Usługi dla luźno powiązanych systemów 875

Buforowanie żądań za pomocą kolejkowanych komponentów 875
Tworzenie prostego kolejkowanego komponentu 878
Obsługa błędów 881
Kolejkowane komponenty, transakcje i korzystanie z puli 884

Tworzenie systemów typu publikuj-subskrybuj za pomocą luźno powiązanych zdarzeń 885
Tworzenie prostego luźno powiązanego zdarzenia 886
Dodawanie stałych subskrypcji do zdarzenia LCE 889
Dodawanie tymczasowych subskrypcji do zdarzenia LCE 892
Zaawansowane zagadnienia związane z LCE 897
Podsumowanie 899
Propozycje dalszych lektur 899

Rozdział 39. Zabezpieczanie komponentów serwerowych 901

Kontrola uwierzytelniania w aplikacjach COM+ 901
Dodawanie zabezpieczeń opartych na rolach w COM+ 904
Przygotowywanie projektu testowego 904
Włączanie zabezpieczeń opartych na rolach 908
Definiowanie ról i przypisywanie do nich członków 908
Autoryzacja użytkowników należących do danej roli za pomocą atrybutu SecurityRole 909
Programowa autoryzacja użytkowników należących do ról za pomocą ContextUtil lub SecurityContext 911
Tworzenie zaufanych podsystemów za pomocą tożsamości 912
Zabezpieczanie kolejkowanych komponentów 913
Zabezpieczanie luźno powiązanych zdarzeń 914
Ukrywanie komponentów za pomocą atrybutu PrivateComponent 914
Podsumowanie 914
Propozycje dalszych lektur 914

Rozdział 40. Zaawansowane programowanie usług Enterprise 915

Programowanie katalogu COM+ 916
Programowanie tymczasowych subskrypcji luźno powiązanych zdarzeń 917
Zamykanie aplikacji COM+ i usuwanie komponentów 919
Łączenie błyskawicznego tworzenia aplikacji i usług Enterprise 922
Podsumowanie 929
Propozycje dalszych lektur 929

Dodatki 931

Skorowidz 933

Rozdział 1.
Wprowadzenie do .NET

W skrócie

W tym rozdziale przedstawiamy, czym jest .NET, jak działa, jak programiści mogą go wykorzystać i opisujemy wiele jego nowych funkcji. Omawiamy tu wszystkie aspekty zastosowania .NET Framework, jego architekturę i różne komponenty.

Mówiąc najprościej, .NET to następna generacja platformy programowej firmy Microsoft przeznaczonej dla oprogramowania dla Windows i internetu. Jednak .NET jest czymś więcej niż tylko nową platformą do tworzenia oprogramowania. Aby Czytelnicy zrozumieli, czym naprawdę jest .NET i jak można go najlepiej wykorzystać, przedstawimy podstawowe cele wprowadzenia platformy .NET. .NET to na nowo zaprojektowana, rewolucyjna architektura programowa firmy Microsoft, której zadaniem jest zrealizowanie nowej myśli przewodniej Microsoftu: „program jako usługa". W firmie Microsoft postanowiono urzeczywistnić tę ideę, a .NET jest pierwszym krokiem w tym kierunku. Jednym z ważnych założeń tej architektury jest właśnie „oprogramowanie jako usługa", a co to oznacza w rzeczywistości, przedstawimy w kolejnych punktach.

Oprogramowanie jako usługa

Zagadnienie oprogramowania jako usługi nie jest tak kłopotliwe i złożone, jak można by sądzić. Wiele firm korzysta z tej technologii, nie zdając sobie z tego sprawy. W firmach tych przekształcono aplikacje Win32 na usługi programowe lub aplikacje te zostały tak zmodyfikowane, aby dały się zintegrować z usługami oferowanymi przez te firmy.

Korzystając z .NET oraz obsługujących tę architekturę technologii (na przykład usług sieciowych), programiści mogą w łatwy sposób tworzyć usługi programowe. Usługi te mogą być następnie publikowane w sieci i udostępniane innym programistom. .NET składa się z trzech kluczowych części. Są to:

- produkty .NET,
- usługi .NET,
- .NET Framework.

W następnych punktach przyjrzymy się kolejno tym elementom.

Produkty .NET

Produkty .NET to wszystkie obecne i przyszłe produkty firmy Microsoft wchodzące w skład .NET. Najważniejszym produktem w tym obszarze jest Visual Studio .NET. Visual Studio .NET to produkt pozwalający programistom na łatwe tworzenie aplikacji oraz oferujący wiele funkcji wspomagających proces tworzenia. IDE Visual Studio umożliwia łatwy dostęp do tych narzędzi.

Usługi .NET

Usługi .NET to usługi oferowane przez firmę Microsoft, które pozwalają programistom na wykorzystanie gotowych zbiorów funkcji we własnych aplikacjach. Obecnie Microsoft oferuje tylko jedną usługę — Passport. Usługa Microsoft Passport pozwala na autoryzację użytkowników i jest wykorzystywana w takich witrynach jak HotMail oraz NineMSN. Usługa Passport pozwala programistom na zrealizowanie scentralizowanej identyfikacji użytkowników.

.NET Framework

Ponieważ .NET Framework jest całkowicie zintegrowany z systemem operacyjnym Windows — i dlatego jest całkowicie przezroczysty dla programisty i użytkownika — pozwala programistom na tworzenie szybkich, elastycznych, skalowalnych i efektywnych aplikacji. Dodatkowo programiści mogą tworzyć aplikacje współdziałające z innymi aplikacjami .NET oraz komponentami napisanymi za pomocą zupełnie innych języków programowania.

Jest to nowy, świetny pomysł, możliwy do zrealizowania tylko dzięki odpowiedniej konstrukcji .NET Framework. W firmie Microsoft takie działanie zaplanowano już od początku. Oznacza to, że można pisać komponenty w dowolnym języku programowania zgodnym z .NET, na przykład C#, i korzystać z nich w aplikacji napisanej w VB.NET. Koncepcje i technologie dodane do .NET Framework obejmują bibliotekę klas .NET Framework, wspólne środowisko uruchomieniowe, Windows Forms (WinForms) oraz ASP.NET. W następnych punktach przyjrzymy się tym elementom nieco dokładniej.

Podstawowa biblioteka klas .NET Framework

Podstawowe klasy .NET Framework to zbiór funkcji, obiektów, właściwości i metod, z których można skorzystać w dowolnym języku zgodnym z .NET. W tej części architektury .NET znajduje się biblioteka ADO.NET, która udostępnia programistom funkcje, obiekty, właściwości i metody pozwalające korzystać z baz danych. W podstawowej bibliotece klas dostępne są również narzędzia pozwalające programistom na korzystanie z XML.



Funkcje XML stanowią jeden z najlepszych składników .NET Framework. Obsługa XML jest naturalna dla tej platformy, a składnia jest bardzo intuicyjna.

Wspólne środowisko uruchomieniowe

Wspólne środowisko uruchomieniowe (ang. *Common Language Runtime — CLR*) jest centralnym elementem .NET Framework i jest odpowiedzialne za uruchamianie aplikacji .NET, zarządzanie nimi, jak również za kompilację aplikacji .NET do postaci macierzystego kodu binarnego. Jest to środowisko, w którym działają aplikacje .NET. Wspólne środowisko uruchomieniowe udostępnia programistom wiele przydatnych i ważnych funkcji, prostsze błyskawiczne tworzenie aplikacji (ang. *Rapid Application Development — RAD*), zarządzanie pamięcią, skalowalność oraz jednolitą integrację różnych programów i języków.

Integracja różnych języków

Możliwości integracji różnych języków są dostępne dzięki doświadczeniu zdobytemu przy okazji tworzenia technologii COM i DCOM. W przypadku COM większość kodu komponentów niskiego poziomu została napisana w C++ (czasami nawet w C), a klasy te były wykorzystywane w VB. W przypadku korzystania z tej technologii często napotykamy na ogromne dziury, ponieważ programowanie w języku C++ znacznie różni się od programowania w VB. Jednak w przypadku .NET tworzenie takich bibliotek w C# i korzystanie z nich w VB.NET (a czasami nawet odwrotnie) jest dziecinnie proste.

Prostsze błyskawiczne tworzenie aplikacji

Pomysł błyskawicznego tworzenia aplikacji nie jest żadną nowością. Ponieważ jednak klasy bazowe .NET Framework udostępniają programistom gotowe komponenty, a dodatkowo można korzystać z tradycyjnych technik wielokrotnego wykorzystania kodu, ilość kodu, jaki musi napisać programista przy tworzeniu aplikacji, została znacznie ograniczona.

Zarządzanie pamięcią

Zarządzanie pamięcią jest jedną z najważniejszych funkcji realizowanych przez wspólne środowisko uruchomieniowe. Działanie tego mechanizmu jest niewidoczne dla programisty, co umożliwia mu skupienie się na tworzeniu aplikacji zamiast na pamiętaniu o różnych aspektach zarządzania pamięcią.

Skalowalność

Zwiększona skalowalność jest bezpośrednim wynikiem omówionych już funkcji, takich jak błyskawiczne tworzenie aplikacji oraz zarządzanie pamięcią. Błyskawiczne tworzenie aplikacji pozwala poświęcić więcej czasu na dostrajanie i poprawianie wydajności aplikacji, natomiast funkcje zarządzania pamięcią dostępne we wspólnym środowisku uruchomieniowym zwiększają wydajność operacji na pamięci oraz ilość zasobów dostępnych dla aplikacji.



Pamiętaj, że można również skonwertować MSIL na kod rodzimy, dzięki czemu otrzymujemy zwykłe pliki wykonywalne, takie jak w przypadku C lub C++.

Integracja różnych języków

Funkcja ta jest prawdopodobnie najczęściej dyskutowaną możliwością .NET. Jest to radykalna i bardzo znacząca zmiana w porównaniu z tradycyjnym podejściem, ponieważ pozwala łączyć ze sobą wszystkie języki obsługujące .NET. Na przykład C++ jest uważany za język o znacznie większych możliwościach niż Visual Basic, ale .NET znacznie wyrównuje te różnice. Wszystkie aplikacje .NET są kompilowane do postaci języka pośredniego (ang. *Intermediate Language — IL*). Język pośredni wyrównuje efektywność, złożoność i optymalizacje z różnych języków .NET. Aby język programowania mógł zostać uznany za zgodny z .NET, kod źródłowy napisany w tym języku musi być kompilowany do postaci języka pośredniego (IL).

W dalszej części rozdziału, w punkcie „Zalety CLR", omówimy dokładniej te możliwości.

Aplikacje korzystające z Windows Forms (WinForms)

WinForms to istotna część systemu tworzenia aplikacji dla Win32 dostępnego w .NET. Jest to dostępny dla wszystkich języków zgodnych z .NET odpowiednik API graficznego oraz formantów. Jest to odpowiednik MFC stosowanego przez programistów C++ oraz Win32 API wykorzystywanego w VB. Dzięki unifikacji oraz integracji ze wspólnym środowiskiem uruchomieniowym biblioteka WinForms udostępnia programistom takie udogodnienia jak projektowanie metodą przeciągnij i upuść, które mogą być wykorzystane do wizualnego tworzenia aplikacji .NET.

ASP.NET

ASP.NET to następna generacja Active Server Page (ASP) — technologii dostępnej w serwerze Internet Information Server (IIS). Największa i najbardziej popularna zmiana w .NET Framework powstała w czasie transformacji ASP na ASP.NET. W porównaniu ze starą, klasyczną wersją ASP ASP.NET jest całkowicie inną platformą. Wiele osób wydaje się gubić w ASP.NET i niektórych jego nowych funkcjach. Trzeba wyjaśnić kilka kwestii. ASP.NET nie korzysta już z VBScript. Obecnie obsługuje wszystkie kompilowane języki .NET, takie jak VB.NET, C# oraz JScript .NET. ASP.NET bazuje na obiektowym paradygmacie sterowanym zdarzeniami, a nie na paradygmacie sekwencyjnym zastosowanym w klasycznym ASP. Oprócz podobnej nazwy ASP oraz ASP.NET mają jeszcze kilka podobieństw.

ASP.NET jest językiem wykonywanym na serwerze, sterowanym zdarzeniami, wykorzystującym dowolny z języków .NET. Ponieważ ASP.NET korzysta z paradygmatu obiektowego, wymuszona została bardziej formalna składnia. Pozwala to znacznie poprawić wydajność i ułatwić wykorzystanie komponentów, zdarzeń, właściwości i metod. W porównaniu z tradycyjnym ASP największą zaletą ASP.NET jest integracja ze wspólnym środowiskiem uruchomieniowym. ASP.NET korzysta z wielu udogodnień wspólnego środowiska uruchomieniowego, w tym obsługi standardowych technik programistycznych. Dodatkowo strony ASP.NET są kompilowane i po pierwszym uruchomieniu buforowane w pamięci serwera. Daje to znaczne przyspieszenie pracy w porównaniu z tradycyjnymi stronami ASP, które są kompilowane za każdym uruchomieniem.

Innym problemem tradycyjnych stron ASP jest skalowalność. W przypadku ASP kod oraz HTML mogą się znajdować w tym samym pliku, co często powoduje, że niemal niemożliwe jest zapanowanie nad kodem i projektem. Oczywiście obiekty COM pozwalały rozdzielić kod na warstwy, ale wielu programistów po prostu umieszczało kod logiki biznesowej bezpośrednio na stronach ASP. To nie było dobre rozwiązanie.

Z wielu powodów ASP.NET jest istotną częścią .NET. Najważniejszym powodem jest to, że strony ASP.NET są superklejem łączącym ze sobą internet i aplikacje Win32. Usługi sieciowe są uważane za część ASP.NET. Jako protokół komunikacyjny w usługach sieciowych wykorzystywany jest Simple Object Access Protocol (SOAP). SOAP jest mocno związany z XML, co sprawia, że jest idealny w erze programowania dla internetu.

Zadaniem usług sieciowych jest udostępnienie innym programistom komponentów i funkcji ASP.NET, a nawet gotowych aplikacji (konfigurowanych poprzez sieć). Usługi sieciowe pozwalają programistom na zachowanie dla siebie sposobu działania kodu aplikacji i udostępnienie tylko minimalnej liczby metod oraz zmiennych składowych pozwalających na zaimplementowanie funkcji oferowanej przez daną usługę. To wszystko jest związane z wizją Microsoftu dotyczącą oprogramowania jako usługi. Usługi sieciowe są niezwykle efektywnym i popularnym dodatkiem do .NET i aby je w pełni docenić, trzeba wypróbować je w działaniu.

Potencjał usług sieciowych

W przypadku usług sieciowych klienci mogą oczekiwać wielu zaawansowanych komponentów internetowych oraz usług, które łatwiej pozwalają spełnić ich oczekiwania. Wyobraźmy sobie świat za kilka lat. Możesz być programistą tworzącym usługę sieciową pomagającą klientom uczyć poprzez sieć swoich nowych pracowników bez żadnej interwencji ze strony człowieka. W przypadku .NET możliwości są nieograniczone i sytuacja może się tylko poprawiać.

Skoro już przedstawiliśmy najważniejsze komponenty .NET Framework, w następnym punkcie omówimy jego zadania.

Zadania .NET Framework

.NET jest najlepszym produktem firmy Microsoft od czasu opracowania systemu MS-DOS w roku 1982. .NET posiada cechy ważne zarówno dla programistów, jak i dla ich klientów. Programiści mają wszystkie narzędzia pozwalające w łatwy sposób tworzyć aplikacje, zarządzać nimi i rozprowadzać je. Mogą lepiej niż do tej pory wykorzystać internet. Wspólne środowisko uruchomieniowe .NET powoduje również, że aplikacje rzadziej będą ulegały awariom i częściej programiści osiągną oczekiwane wyniki: czystą, szybką oraz solidną implementację zapewniającą długie czasy nieprzerwanego działania.

.NET Framework jest zaprojektowany w taki sposób, aby osiągnąć następujące cele:

- Spójne środowisko programowania obiektowego, w którym kod może być zapisywany i wykonywany lokalnie, wykonywany lokalnie, ale rozprowadzany w internecie lub wykonywany zdalnie.

- Środowisko wykonywania kodu, które minimalizuje konflikty przy rozprowadzaniu oprogramowania i wersjonowaniu.
- Środowisko wykonywania kodu, które gwarantuje bezpieczne wykonywanie kodu oraz kontrolę kodu napisanego przez nieznanych lub nie w pełni zaufanych autorów.
- Środowisko wykonywania kodu, które eliminuje problemy z wydajnością skryptów lub języków interpretowanych.
- Spójność środowiska dla programistów pracujących nad różnymi typami aplikacji, na przykład aplikacjami Windows oraz aplikacjami WWW.
- Komunikacja zbudowana na bazie standardów przemysłowych, co zapewnia, że .NET Framework może integrować się z innymi produktami. Na przykład w firmie Borland utworzono bibliotekę komponentów .NET współpracujących z komponentami Java; są one do wykorzystania w programie C# Builder.

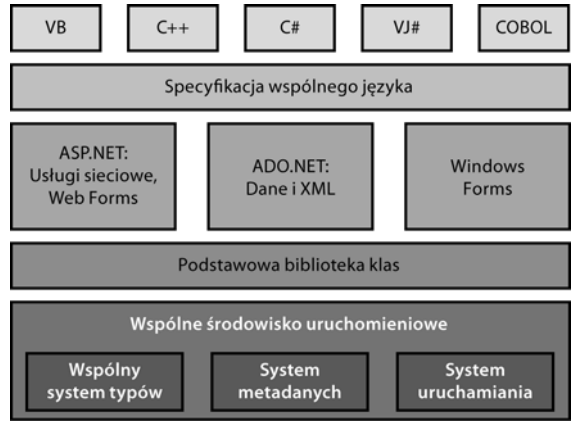
W następnym punkcie omówimy architekturę .NET Framework oraz sposób, w jaki zapewnia ona wszystkie przedstawione tu cele.

Poznajemy architekturę .NET Framework

.NET Framework składa się z:

- biblioteki klas .NET Framework;
- wspólnego środowiska uruchomieniowego.

Biblioteka klas .NET Framework udostępnia typy (CTS), które są wykorzystywane we wszystkich językach .NET. Wspólne środowisko uruchomieniowe zawiera moduł ładowania klas, który wczytuje kod programu zapisany w postaci IL. Następnie kod IL jest kompilowany do postaci kodu maszynowego komputera, uruchamiany i kontrolowany w celu zapewnienia odpowiedniego poziomu bezpieczeństwa oraz zapewnienia obsługi wątków. Na rysunku 1.1 przedstawiona jest architektura .NET Framework i jego różnych składników.



Rysunek 1.1. *Architektura .NET Framework*

Na górze schematu architektury znajdują się języki programowania, takie jak VB.NET, C#, VJ# oraz VC++ .NET; programiści mogą tworzyć aplikacje (używając dowolnego z tych języków), korzystając z Windows Forms, Web Forms, usług Windows oraz usług sieciowych XML. Dolne dwie warstwy zawierają bibliotekę klas .NET Framework oraz wspólne środowisko uruchomieniowe.

Poznajemy zadania .NET Framework

Jak już wspomnieliśmy, .NET Framework zawiera dwa główne składniki: wspólne środowisko uruchomieniowe oraz bibliotekę klas .NET Framework. Wspólne środowisko uruchomieniowe jest podstawą dla .NET Framework. Działa ono podobnie do agenta zarządzającego kodem w czasie jego działania, udostępniając podstawowe usługi, takie jak zarządzanie pamięcią, wątkami czy wywołania zdalne. Obsługuje również ścisłą kontrolę typów, co poprawia jakość kodu i zwiększa bezpieczeństwo i odporność aplikacji. Koncepcja zarządzania kodem jest podstawową zasadą wspólnego środowiska uruchomieniowego. Kod, który ma być wykonywany przez wspólne środowisko uruchomieniowe, jest nazywany kodem zarządzanym, natomiast kod, który nie był projektowany do wykonania przez wspólne środowisko uruchomieniowe, jest nazywany kodem niezarządzanym.

Biblioteka klas, która jest integralną częścią .NET Framework, jest zbiorem klas lub typów do wielokrotnego wykorzystania, używanym do tworzenia aplikacji różnych rodzajów, od tradycyjnych aplikacji wiersza poleceń, do dowolnych aplikacji z graficznym interfejsem użytkownika, na przykład korzystających z Windows Forms lub ASP.NET Web Forms, jak również nowych usług sieciowych XML.

Kod napisany w języku zgodnym z CLS powinien być zgodny z kodem napisanym w innym języku zgodnym z CLS, ponieważ kod napisany w tych językach jest kompilowany do kodu pośredniego (IL), a dopiero maszyna wspólnego środowiska uruchomieniowego wykonuje kod IL. Zapewnia to współdziałanie ze sobą języków zgodnych z CLS. Microsoft .NET Framework obsługuje takie języki jak: Microsoft Visual Basic .NET, Microsoft Visual C#, Microsoft Visual C++ .NET oraz Microsoft Visual J# .NET.

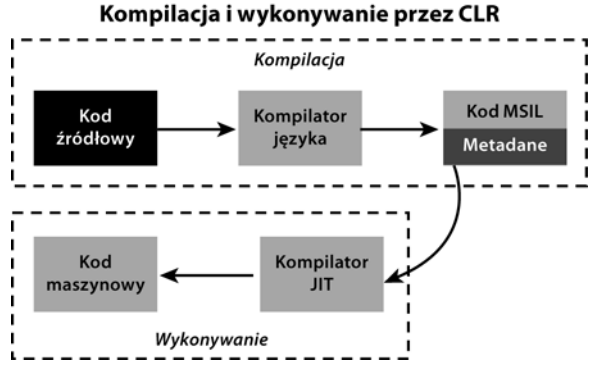
Kompilatory tych języków generują kod pośredni nazywany Microsoft Intermediate Language (MSIL), który zapewnia możliwość współdziałania ze sobą programów .NET napisanych w różnych językach.

Standard *Common Language System* (CLS) definiuje infrastrukturę potrzebną do uruchomienia kodu IL. W CLI (ang. *Common Language Infrastructure* — infrastruktura wspólnego języka) dostępny jest wspólny system typów (ang. *Common Type System — CTS*) oraz takie usługi jak bezpieczeństwo typów, wykonywanie kodu zarządzanego oraz wykonywanie wersji kodu wybranej spośród kilku zainstalowanych w tym samym czasie.

Co to jest MSIL?

MSIL można zdefiniować jako niezależny od procesora zbiór instrukcji wykorzystywanych przez skompilowane aplikacje .NET. Zawiera on instrukcje do ładowania, zapisywania i inicjalizacji obiektów. Kopiując aplikację napisaną w C# lub dowolnym innym języku zgodnym z CLS, otrzymujemy w wyniku kod MSIL. Gdy wspólne środowisko uruchomieniowe uruchamia aplikację, kompilator Just-In-Time (JIT) konwertuje MSIL na kod maszynowy zrozumiały dla procesora danego komputera. Proces ten jest przedstawiony na rysunku 1.2.

Rysunek 1.2. *Konwersja MSIL na kod maszynowy*



Aby możliwa była integracja różnych języków, MSIL współdziała z metadanymi oraz wspólnym systemem typów. Jak już wspominaliśmy, wspólny system typów jest jednym z podstawowych mechanizmów.NET Framework, który zapewnia integrację różnych języków przez obsługę typów i operacji używanych w większości języków programowania.

Co to jest plik Portable Executable (PE)?

Gdy kompilujemy naszą aplikację, oprócz kodu MSIL kompilator generuje metadane będące zbiorem informacji o aplikacji. Te metadane są zapisywane w postaci binarnej w przenośnym pliku wykonywalnym. Microsoft zaprojektował format przenośnego pliku wykonywalnego (PE) na potrzeby wszystkich systemów bazujących na Win32. Zastosowanie przenośnego pliku wykonywalnego przy uruchamianiu programu .NET jest następujące:

- Kod źródłowy jest tworzony w dowolnym języku .NET.
- Następnie kod ten jest kompilowany za pomocą odpowiedniego kompilatora języka .NET.
- Kompilator generuje kod MSIL oraz manifest jako przeznaczoną tylko do odczytu część pliku EXE mającego standardowy nagłówek PE (plik wykonywalny Win32).
- Wykonując aplikację, system operacyjny ładuje PE oraz wszystkie podrzędne biblioteki ładowane dynamicznie.
- Następnie system operacyjny rozpoczyna wykonywanie kodu MSIL umieszczonego wewnątrz pliku PE. Ponieważ kod MSIL nie jest dla procesora formatem

wykonywalnym, nie może być uruchomiony bezpośrednio. Wspólne środowisko uruchomieniowe kompiluje na bieżąco MSIL na kod maszynowy, wykorzystując do tego celu kompilator JIT.

Poznajemy kompilator JIT

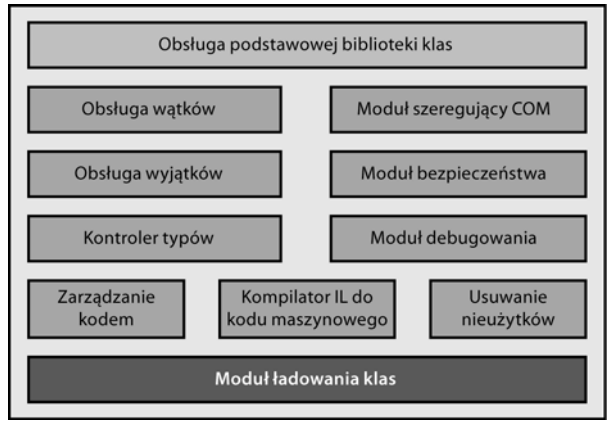
Jak już wcześniej pisaliśmy, kompilator JIT jest integralną częścią wspólnego środowiska uruchomieniowego. Kompiluje on kod MSIL na kod maszynowy i wykonuje generowany na bieżąco kod, który po wykonaniu jest buforowany do późniejszego wykorzystania. Następnie zapamiętany w buforze kod jest używany przy wykonywaniu kolejnego żądania.

Wspólne środowisko uruchomieniowe

Podstawą platformy .NET jest wspólne środowisko uruchomieniowe, które jest odpowiedzialne za zarządzanie wykonywaniem kodu przeznaczonego na platformę .NET. Kod, który do uruchomienia wymaga zastosowania wspólnego środowiska uruchomieniowego, jest nazywany kodem zarządzanym. Kod zarządzany wymaga dostępności podstawowego zbioru usług (które omówimy za chwilę) udostępnianych przez wspólne środowisko uruchomieniowe. Usługi oferowane przez wspólne środowisko uruchomieniowe są wymienione w następnym punkcie.

Składniki CLR

Na rysunku 1.3 przedstawiona jest architektura wspólnego środowiska uruchomieniowego i jego różnych składników.



Rysunek 1.3. *Architektura wspólnego środowiska uruchomieniowego .NET*

Na poniższej liście wszystkie te składniki są opisane nieco bardziej szczegółowo.

- **Moduł ładowania klas** — jest odpowiedzialny za ładowanie klas do wspólnego środowiska uruchomieniowego.
- **Kompilatory MSIL do kodu maszynowego** — konwertują kod MSIL na zależny od komputera kod maszynowy.
- **Zarządzanie kodem** — zarządza kodem w czasie jego wykonywania.
- **Zarządzanie pamięcią i usuwanie nieużytków** — realizują automatyczne zarządzanie pamięcią.
- **Moduł bezpieczeństwa** — wymusza ograniczenia nakładane przez zabezpieczenia dostępu do kodu. Zabezpieczenia na poziomie systemu można konfigurować za pomocą narzędzi .NET, które zawiera *Panel sterowania*.
- **Kontroler typów** — zapewnia dokładną kontrolę typów.
- **Obsługa wątków** — komponent ten zapewnia obsługę wielowątkowości w aplikacjach.
- **Obsługa wyjątków** — zapewnia mechanizm obsługi wyjątków w kodzie za pomocą bloków `try..catch..finally`.
- **Moduł debugowania** — pozwala programistom na debugowanie różnych typów aplikacji.
- **Moduł szeregujący COM** — pozwala aplikacjom .NET na wymianę danych z aplikacjami COM.
- **Obsługa podstawowej biblioteki klas** — udostępnia potrzebne aplikacji klasy (typy).

Biblioteka klas .NET Framework

Jak można wywnioskować z nazwy, biblioteka klas jest zbiorem klas i związanych z nimi struktur, które mogą być wykorzystane jako podstawowe elementy budulcowe aplikacji. Można więc bezpiecznie przyjąć, że ten zbiór klas jest pewnego rodzaju API. Są one granicznym interfejsem pomiędzy aplikacją i systemem operacyjnym. Taka koncepcja nie jest niczym nowym dla programistów używających języka Visual Basic — biblioteka ADO, API Win32 oraz biblioteka usług COM+ pozwalają na korzystanie z istniejącego kodu w aplikacjach. Biblioteka klas jest olbrzymią biblioteką kodu, który można wykorzystać jako podstawę dla funkcji aplikacji.

Biblioteka .NET Framework zawiera klasy, interfejsy i typy wartości, które nie tylko zapewniają dostęp do mechanizmów systemu, ale również optymalizują proces tworzenia aplikacji. Aby zrealizować współpracę między językami, typy w .NET Framework są zgodne z CLS i dzięki temu mogą być wykorzystywane przez dowolne języki programowania, których kompilatory generują kod zgodny z Common Language System.

Klasy zebrane w .NET Framework są fundamentami, na których zbudowane zostały aplikacje, komponenty i formanty .NET. W .NET Framework znajdują się typy pozwalające na wykonanie następujących operacji:

- Reprezentacja podstawowych typów danych i wyjątków.
- Hermetyzacja struktur danych.
- Wykonywanie operacji wejścia-wyjścia.
- Dostęp do informacji o załadowanych typach.
- Wywoływanie mechanizmów kontroli zabezpieczeń z .NET.
- Zapewnienie dostępu do danych, bogate GUI klienckie oraz GUI klienckie kontrolowane przez serwer.

Odwoływanie się do istniejących bibliotek kodu w Visual Basicu jest tradycyjnie bardzo proste. Jednak często biblioteki kodu nie były od razu dostępne dla programistów korzystających z języka Visual Basic. Programiści C++ mogli korzystać z wszystkich funkcji nowych bibliotek, natomiast programiści VB musieli poczekać, aż Microsoft lub inna firma napisze interfejs pozwalający na wykorzystanie biblioteki z poziomu VB. Równie często występowały problemy z dokumentacją. Biblioteka .NET Framework pomaga rozwiązać te problemy przez udostępnienie jednolitego interfejsu dla wszystkich języków korzystających z .NET Framework. Również organizacja klas w .NET Framework znacznie ułatwia pracę.

Sposób organizacji klas w bibliotece .NET Framework

Podstawową jednostką organizacyjną w bibliotece klas jest przestrzeń nazw. Przestrzeń nazw jest po prostu pojemnikiem na funkcje — opisuje ona grupę klas i konstrukcji o podobnym przeznaczeniu. Koncepcję przestrzeni nazw można porównać do katalogów w systemie plików — oba mechanizmy pozwalają na zaimplementowanie organizacji rodzic-dziecko w zbiorze obiektów. Podobnie jak katalog może zawierać inne katalogi i dokumenty, przestrzeń nazw może zawierać inne przestrzenie nazw, klasy, delegacje i tak dalej.

Wszystkie przestrzenie nazw mają wspólny korzeń — przestrzeń `System`. Przestrzeń nazw `System` jest jednym jedynym korzeniem dla wszystkich pozostałych przestrzeni. Na przykład przestrzeń nazw `System` zawiera struktury definiujące wspólne typy danych wykorzystywane w .NET, takie jak `Boolean`, `DateTime` oraz `Int32`. Zawiera ona najważniejszy typ danych — `Object` — który jest podstawową klasą, po której dziedziczą wszystkie inne klasy .NET.

Korzyści ze stosowania biblioteki klas .NET Framework

Do tej pory opisaliśmy strukturę .NET Framework i jego składniki. W kolejnych punktach zostaną wymienione niektóre korzyści płynące ze stosowania biblioteki klas .NET Framework.

Zwiększanie wydajności programistów

Ważnym celem projektu biblioteki klas było zwiększenie wydajności programistów. Być może jesteś zaskoczony tym, że głównym celem przy projektowaniu biblioteki klas było zwiększenie produktywności nie przez wprowadzenie nowych funkcji, ale przez

przepakowanie istniejących funkcji na postać obiektową. Faktycznie, choć biblioteka klas zawiera kilka nowych funkcji, większość funkcji udostępnianych w przestrzeniach nazw była już wcześniej dostępna w Win32 API, formantach ActiveX, bibliotekach DLL bazujących na COM i tak dalej. Teraz są one upakowane w taki sposób, że można programować je na wyższym poziomie abstrakcji, na którym nie trzeba się zajmować problemami obsługi niewielkich struktur danych wymaganych przez Win32 API. Bezpośrednim wynikiem tej operacji jest to, że w swojej aplikacji należy obsługiwać tylko jeden, nadrzędny wzorzec projektowy — komponenty są wykorzystywane w środowisku obiektowym. Co więcej, taki model pracy jest dostępny we wszystkich językach przeznaczonych dla wspólnego środowiska uruchomieniowego.

Na wypadek, gdybyś miał wątpliwości, dodajmy, że biblioteka klas .NET Framework nie zastąpiła Win32 API. Biblioteka klas nadal korzysta z istniejących klas Win32 API, które komunikują się z Win32 API i wykonują kod poprzez mechanizm nazywany P/Invoke.

Wyszukiwanie potrzebnego kodu

Jeżeli zaczniemy analizować bibliotekę klas pod kątem zwiększenia wydajności programowania, natychmiast zauważymy kilka zalet. Po pierwsze, biblioteka klas pozwala łatwo wyszukiwać potrzebne funkcje. Rozmiar biblioteki może początkowo wydawać się przytłaczający, ale logiczny podział wewnątrz przestrzeni tabel pozwala łatwo lokalizować odpowiednie zakresy funkcji. Na przykład przestrzeń nazw `System.Drawing` zawiera klasy oferujące funkcje rysunkowe. Przestrzeń nazw `System.IO` udostępnia podstawowe operacje wejścia-wyjścia i tak dalej. Wewnątrz przestrzeni nazw `System.Drawing` znajdują się obiekty dobrze znane każdemu programiście Windows: `Pen`, `Brush` itp.

W przypadku każdego API o znacznym rozmiarze zadanie lokalizacji kodu lub funkcji potrzebnej do wykonania określonego zadania jest sporym problemem. Porównajmy przedstawioną właśnie organizację przestrzeni tabel z płaską, monolityczną przestrzenią nazw dostępną w Win32 API. Głównym problemem jest to, że wraz z rozwojem rozmiaru Win32 API jego programiści musieli być coraz bardziej kreatywni przy wymyślaniu nazw funkcji. Możliwość określenia zadania funkcji tylko na podstawie jej nazwy stawała się coraz bardziej ograniczona. Szczerze mówiąc, możliwe jest osiągnięcie wysokiej wydajności przy wykorzystywaniu Win32 API w Visual Basicu. Wymaga to jednak dużej determinacji i dostępu do wielu informacji referencyjnych. Biblioteka klas .NET Framework jest bardziej przyjaznym API — szukane rzeczy są tam, gdzie tego oczekujemy.

Kod działający pod kontrolą środowiska uruchomieniowego .NET jest nazywany kodem zarządzanym. Kod niezarządzany to cały kod działający poza środowiskiem uruchomieniowym .NET, na przykład Win32 API, komponenty COM lub formanty ActiveX. Obecnie jedynym narzędziem firmy Microsoft pozwalającym pisać kod niezarządzany jest Visual C++.

Wykorzystanie biblioteki klas

Po wykonaniu pierwszego kroku, jakim jest wyszukanie właściwej klasy, kolejnym jest jej wykorzystanie, co również jest bardzo proste w bibliotece klas .NET. Poprzez udostępnienie funkcji poprzez właściwości i metody mamy łatwy dostęp do funkcji, które mogą wykonywać niezmiernie skomplikowane operacje. Jest to działanie w duchu ukrywania

i hermetyzacji danych — operujemy na czarnych skrzynkach i nie musimy wiedzieć, co dzieje się wewnątrz skrzynki ani tym bardziej się tym przejmować. Musimy jedynie zapewnić odpowiednią komunikację z interfejsem tej czarnej skrzynki.

Jest to dosyć ważny element produktywności. Porównajmy to do nakładu pracy przy bezpośrednim wykorzystaniu Win32 API. Ponieważ Visual Basic pozwalał na korzystanie z API mającego zupełnie inny system typów, nie było jasnego odwzorowania pomiędzy wywołaniami API a składnią języka Visual Basic. Obecnie, dzięki roli, jaką pełni wspólny system typów, wszystkie języki komunikują się z komponentami uruchomieniowymi z wykorzystaniem tych samych typów danych. Dzięki temu klasy z biblioteki klas .NET mogą być w ten sam sposób wykorzystywane przez każdy język .NET. Po raz pierwszy programiści używający języka Visual Basic mają pełny dostęp do wszystkich funkcji biblioteki, bez żadnego spadku produktywności.

Przejście pomiędzy funkcjami charakterystycznymi dla języka i biblioteką klas jest teraz całkowicie płynne. Inaczej mówiąc, programiści w końcu mogą tworzyć swoje aplikacje bez ciągłego skakania po różnych paradygmatach kodowania (na przykład programowania proceduralnego w Win32 API i programowania ukierunkowanego na komponenty w VB). Wzorce projektowe mogą pozostać spójne w całym kodzie, niezależnie od właśnie wykorzystywanych komponentów. Osiągnięto to dzięki zbudowaniu całej biblioteki w sposób wspierający programowanie obiektowe i ukierunkowane na komponenty. Co więcej, z biblioteki klas korzysta się w naturalny dla programistów VB sposób. Nie jest wymagana żadna ezoteryczna składnia, po prostu odwołujemy się do komponentu, tworzymy jego egzemplarz, a następnie korzystamy z jego metod i właściwości.

Ponieważ biblioteka klas .NET jest napisana w kodzie zarządzanym działającym wewnątrz wspólnego środowiska uruchomieniowego, korzysta ona z wszystkich zalet wykorzystania kodu zarządzanego:

- uwolnienia od identyfikatorów GUID, uchwytów okien, zmiennych `hResult` i tak dalej;
- automatycznego odzyskiwania pamięci zajętej przez nieużywane zasoby;
- strukturalnej obsługi wyjątków.

Zalety CLR

Wspólne środowisko uruchomieniowe zapewnia wiele usług dla kodu zarządzanego, w tym integrację wielu języków, zwiększone bezpieczeństwo, obsługę wersjonowania i wdrażania, debugowanie i profilowanie oraz zarządzanie pamięcią poprzez automatyczne odzyskiwanie nieużytków. W tym punkcie przedstawimy te zalety nieco bardziej szczegółowo.

Integracja różnych języków

Wspólne środowisko uruchomieniowe pozwala na bezproblemową integrację kodu zarządzanego napisanego w jednym języku z kodem napisanym w innym języku. Jest to możliwe dzięki wykorzystaniu CLS, w którym zdefiniowane są reguły, jakie musi spełniać

każdy język dostępny w .NET Framework. Lista funkcji możliwych do integracji zawiera dziedziczenie, obsługę wyjątków oraz szeregowanie. Na przykład można zdefiniować interfejs w języku COBOL i implementować go w takich językach jak VB.NET i C#.

Zwiększone bezpieczeństwo

Obecnie kod wykonywany na naszych komputerach to nie tylko instalowane przez nas aplikacje, ale również kod pobierany z internetu poprzez strony WWW lub pocztę elektroniczną. Ostatnie przypadki pokazały, że może być to bardzo szkodliwe dla systemu. W jaki sposób .NET pozwala ograniczyć te zagrożenia?

Środowisko .NET posiada zabezpieczenia dostępu do kodu, które pozwalają kontrolować dostęp do chronionych zasobów i operacji. Kod posiada różny poziom zaufania, w zależności od jego pochodzenia i sposobu, w jaki do nas dotarł. Niektóre funkcje tego systemu bezpieczeństwa są wymienione poniżej:

- Administratorzy mogą definiować polityki bezpieczeństwa i przydzielać określone uprawnienia do zdefiniowanych grup kodu.
- Kod może wymagać specjalnych uprawnień od wywołującego go.
- Kod może wymagać określonych uprawnień do jego uruchomienia, innych do wykonywania pewnych funkcji, jak również może wymagać, aby wywołujący go nie miał określonego zbioru uprawnień.

Obsługa wersjonowania i wdrażania

Wspólne środowisko uruchomieniowe obsługuje możliwość jednoczesnego wykonywania wielu wersji tego samego komponentu, nawet w tym samym procesie. Aplikacje przeznaczone dla .NET Framework mogą być instalowane w systemie z użyciem tylko tzw. metody XCOPY, bez zmiany stanu systemu. Użycie metody XCOPY pozwala na skopiowanie plików do katalogu *bin* aplikacji, po czym aplikacja może być natychmiast uruchomiona.

Jest to możliwe, ponieważ kompilatory przeznaczone dla .NET Framework dołączają identyfikatory lub metadane do skompilowanych modułów i wspólne środowisko uruchomieniowe może skorzystać z tych danych w celu załadowania właściwych wersji podzespołów. Identyfikatory zawierają wszystkie informacje potrzebne do załadowania i uruchomienia modułów, jak również zlokalizowania innych modułów wykorzystywanych przez ten podzespół.

Usługi debugowania i profilowania

Wspólne środowisko uruchomieniowe zapewnia funkcje niezbędne do debugowania i profilowania kodu przez programistów. Po dołączeniu działającego programu .NET do debugera można wykonywać takie operacje jak przeglądanie stosu kodu zarządzanego, sprawdzanie zależności i zawartości obiektów. Funkcje te są dostępne dla wszystkich aplikacji, niezależnie od języka, w którym zostały napisane.

Odzyskiwanie nieużytków

W kodzie niezarządzanym bardzo często występują wycieki pamięci. Wyciek pamięci występuje wtedy, gdy pamięć przydzielona w aplikacji nie jest zwalniana w momencie, gdy nie jest ona już potrzebna. Najbardziej narażone na wycieki pamięci są aplikacje pisane za pomocą takich języków jak C++, w których programiści mogą przydzielać pamięć ze sterty. Jednak w kodzie zarządzanym za usuwanie obiektów, z których aplikacja już nie korzysta, odpowiada mechanizm zbierania nieużytków (GC). Gdy w aplikacji napisanej w kodzie zarządzanym przydzielana jest pamięć, jest ona zawsze kojarzona z obiektem. Mechanizm GC zajmuje się zbieraniem tych obiektów, do których nie istnieją już odwołania w aplikacji, dzięki czemu w kodzie zarządzanym nie spotyka się wycieków pamięci. Do identyfikacji obiektów, które powinny być skasowane, GC korzysta z procesu nazywanego eliminacją.

W czasie działania procesu eliminacji GC korzysta z danych, jakie wspólne środowisko uruchomieniowe utrzymuje na temat każdej działającej aplikacji. Na początek GC odczytuje listę głównych obiektów, do których bezpośrednio odwołują się aplikacje. Następnie tworzy listę obiektów, do których odwołują się główne obiekty. Gdy GC zidentyfikuje wszystkie obiekty, do których aplikacja odwołuje się pośrednio lub bezpośrednio, wszystkie pozostałe obiekty w zarządzanej stercie mogą zostać skasowane. Proces zbierania nieużytków jest wywoływany automatycznie przez wspólne środowisko uruchomieniowe, jak również aplikacje mogą jawnie go wywołać poprzez użycie metody `GC.Collect`.

Bogaty model obiektowy oraz biblioteki klas

Wspólne środowisko uruchomieniowe oferuje bogaty zbiór modeli obiektowych oraz bibliotek klas, które udostępniają ogromną liczbę funkcji pozwalających wykonywać takie operacje jak dostęp do relacyjnej bazy danych, wykonywanie operacji wejścia-wyjścia, obsługa danych XML, przeglądanie metadanych i tak dalej. Ponieważ wszystkie modele obiektowe działają na poziomie środowiska uruchomieniowego, łatwo można zaprojektować narzędzia działające we wszystkich językach wspólnego środowiska uruchomieniowego. Podstawową przestrzenią nazw .NET Framework, w której dostępna jest większość funkcji, jest przestrzeń `System`. Przestrzenie nazw są nową koncepcją wprowadzoną do .NET, pozwalającą na logiczne grupowanie związanych klas w pojedynczą jednostkę. Wszystkie klasy wewnątrz przestrzeni nazw muszą mieć unikalne nazwy.

Do tej pory przedstawiliśmy architekturę wspólnego środowiska uruchomieniowego oraz jego różne komponenty. W następnym podrozdziale opiszemy, czym jest podzespół i jakie są różne typy podzespołów.

Co to jest podzespół?

Aby .NET mógł zrealizować zapewnienia o zrewolucjonizowaniu przyszłości informatyki, wewnątrz środowiska musi mieć zaimplementowane wszystkie innowacyjne funkcje. Jedną z tych innowacji jest sposób wykonywania i wersjonowania aplikacji — przeszedł on długą drogę w kierunku eliminowania niesławnego problemu piekła DLL. Microsoft

próbował rozwiązać ten problem, wprowadzając podzespół, który jest podstawową jednostką w aplikacjach .NET.

W aplikacjach .NET podzespoły są wykorzystywane bardzo intensywnie. Są one podstawowymi elementami budulcowymi aplikacji .NET. Reprezentują logiczny zbiór jednego lub więcej programów wykonywalnych lub bibliotek dołączanych dynamicznie (DLL), które zawierają kod aplikacji oraz zasoby. Podzespoły zapewniają formalną strukturę dla widoczności i wersjonowania w czasie działania. Podzespoły zawierają tzw. *manifest*, który jest opisem kodu i zasobów zawartych wewnątrz podzespołu. Ponieważ manifest zawiera wszystkie informacje na temat podzespołu, jest on odpowiedzialny za samoopisującą naturę podzespołu. Zanim zajmiemy się różnymi typami podzespołów i ich zaletami, przedstawimy strukturę podzespołu.

Struktura podzespołu

Ogólna struktura podzespołu jest przedstawiona na rysunku 1.4.

Rysunek 1.4.
Struktura podzespołu



Jak widać, podzespół składa się z czterech elementów. Są to:

- metadane lub manifest;
- dane typów lub metadane opisujące typy;
- moduł;
- zbiór zasobów.

Z tych czterech wymienionych elementów jedynie manifest musi znajdować się w każdym podzespole. Wszystkie pozostałe elementy, takie jak typy i zasoby, są opcjonalne i są wymagane tylko wtedy, gdy są potrzebne do zrealizowania funkcji podzespołu. W następnych punktach opiszemy dokładniej poszczególne składniki podzespołu.

Manifest

Dzięki manifestowi podzespół jest samoopisujący. Manifest zawiera:

- **Identyfikator** — składa się z nazwy, numeru wersji, opcjonalnego kodu lokalizacji i podpisu cyfrowego (jeżeli podzespół ma być współdzielony w wielu aplikacjach).
- **Listę plików** — manifest zawiera listę wszystkich plików składających się na podzespół. Zawiera on również dodatkowe informacje na temat każdego z plików podzespołu, w tym jego nazwę i odcisk kryptograficzny zawartości w czasie

tworzenia manifestu. Odcisk ten jest weryfikowany w czasie uruchamiania w celu sprawdzenia, czy moduł jest spójny i nie został podmieniony. Zależności między podzespołami są zapisane w manifeście podzespołu wywołującego. Dane zależności zawierają numer wersji, który jest wykorzystywany w czasie działania aplikacji do załadowania odpowiedniej wersji zależnego modułu.

- **Typy i zasoby** — jest to lista typów i zasobów udostępnianych przez podzespół. Zawiera ona również informacje o tym, czy typ jest widoczny dla innych podzespołów (udostępniony), czy też prywatny dla aplikacji.
- **Uprawnienia** — podzespoły są modułami, w których wykorzystywane są uprawnienia dostępu do kodu.

Moduł

Modułem może być DLL lub plik wykonywalny w formacie Windows PE. Moduły te zawierają kod kompilowany do postaci Microsoft Intermediate Language (MSIL).



Moduł jest zapisywany w rozszerzonej postaci formatu PE. Jest to niezbędne, ponieważ podstawowy format PE nie pozwala na zapisanie informacji o wersji ani metadanych.

Metadane typów

Typ jest połączeniem danych oraz metod operujących na tych danych. Składa się z właściwości, pól oraz metod. Pola są zmiennymi publicznymi, do których można odwoływać się w bezpośredni sposób. Właściwości są podobne do pól, ale mają skojarzone instrukcje, które są wykonywane przy dostępie do właściwości. Metody reprezentują akcje lub zachowanie się typu.

Wykorzystanie metody XCOPY do bezobsługowej instalacji podzespołów

Jednym z podstawowych zadań .NET Framework jest uproszczenie instalacji poprzez umożliwienie tak zwanego rozmieszczania metodą `XCOPY`. Zanim przedstawimy, w jaki sposób .NET pozwala na rozmieszczanie metodą `XCOPY`, na początek opiszemy, na czym polega ta metoda. Przed wprowadzeniem .NET instalowanie komponentu wymagało skopiowania plików do odpowiednich katalogów, utworzenia wpisów w rejestrze i tak dalej. Obecnie w .NET instalacja komponentu polega na skopiowaniu podzespołu do katalogu *bin* aplikacji klienckiej; dzięki samoopisującej naturze podzespołów aplikacja od razu może być uruchomiona. Jest to nazywane instalacją bezinwazyjną, ponieważ nie zmieniamy konfiguracji komputera przez dokonywanie wpisów w rejestrze.



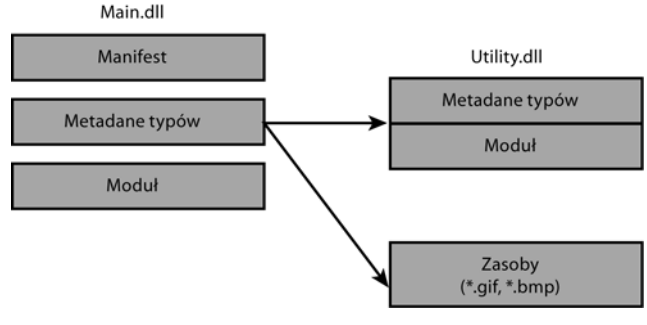
Sama instalacja Microsoft .NET jest implementacją techniki instalacji równoległej. Jeżeli na przykład zainstalujemy .NET Framework w wersji 1.0 i 1.1, okaże się, że powstały dwa katalogi, dla .NET 1.0 oraz .NET 1.1: *C:\WINNT\Microsoft.NET\Framework\v1.0.3705* oraz *C:\WINNT\Microsoft.NET\Framework\v1.1.4322*.

Problemy przy projektowaniu podzespołu

Podzespół może być pojedynczym plikiem zawierającym wszystkie składniki, w tym manifest, metadane typów, kod IL oraz zasoby. Jeżeli podczas budowania aplikacji będziemy korzystać z takich narzędzi jak Visual Studio .NET, każdy projekt najprawdopodobniej będzie odpowiadał jednemu podzespołowi. Można również umieścić zawartość podzespołu w wielu plikach — w takim przypadku podzespół składa się z wielu plików znajdujących się w tym samym katalogu. Manifest podzespołu znajduje się w jednym z plików wykonywalnych lub DLL. Warto pamiętać, że w przypadku podzespołu wieloplikowego system plików nie przechowuje plików w jednym miejscu. Jedynym wyznacznikiem tego, że dane pliki są składnikiem podzespołu, jest to, że są wymienione w manifeście. Za pomocą programów komend wiersza pleceń, takich jak *al.exe*, można dodawać lub usuwać pliki z podzespołu wieloplikowego.

Na rysunku 1.5 zawartość podzespołu `Main` została podzielona na trzy różne pliki. Kod użytkowy znajduje się w osobnym pliku DLL, a duży plik zasobów jest przechowywany w swojej oryginalnej lokalizacji. Jedną z zalet takiego podejścia jest znaczne przyspieszenie ładowania kodu. Środowisko .NET Framework ładuje pliki przy pierwszym odwołaniu, jeśli więc podzespół zawiera kod lub zasoby, do których nie odwołuje się zbyt często, podział pliku na części poprawia szybkość ładowania.

Rysunek 1.5.
Podzespół wieloplikowy



Typy podzespołów

Przy określaniu typu podzespołu powinno się wziąć pod uwagę dwa czynniki:

- **Jak jest wykorzystywany podzespół** — czy jest prywatny dla aplikacji, czy współdzielony przez wiele aplikacji.
- **Jak jest tworzony podzespół** — czy podzespół jest tworzony podczas projektowania, czy w trakcie działania programu.

Bazując na sposobie ich wykorzystywania, podzespoły można podzielić na podzespoły prywatne i współdzielone. Bazując na sposobie tworzenia podzespołów (w czasie projektowania lub działania programu), można zaklasyfikować je jako podzespoły statyczne lub dynamiczne. W poniższych punktach przedstawimy te typy podzespołów nieco bardziej szczegółowo.

Podzespoły prywatne

Podzespół prywatny jest widoczny tylko dla jednej aplikacji. Większość aplikacji .NET jest budowana w postaci prywatnych podzespołów, ponieważ jednym z najważniejszych zadań .NET Framework jest izolacja aplikacji od zmian, jakie inne aplikacje wprowadziły do systemu. Tworząc prywatne podzespoły, zapisujemy je w katalogu *bin* aplikacji, w której są wykorzystywane. Ponieważ zasięg podzespołów prywatnych ogranicza się do bieżącej aplikacji, wymagania co do nazewnictwa podzespołów prywatnych są proste: nazwy muszą być unikalne wewnątrz aplikacji. Każdy prywatny podzespół jest specyficzny dla jednej, określonej aplikacji, więc nie ma potrzeby zapisywać w nich informacji o wersji. Gdy wspólne środowisko uruchomieniowe otrzyma żądanie załadowania podzespołu, przekształca nazwę podzespołu na nazwę pliku zawierającego manifest. Proces ten jest nazywany sondowaniem.

Podzespoły współdzielone

Innym rodzajem podzespołu jest podzespół współdzielony, który jest wykorzystywany przez wiele aplikacji na tej samej maszynie. Podzespoły współdzielone są przechowywane w globalnym buforze podzespołów (ang. *Global Assembly Cache*), który jest centralnym repozytorium podzespołów współdzielonych z danego komputera. Należy pamiętać, że w .NET to programiści podejmują decyzję o współdzieleniu kodu pomiędzy aplikacjami.

Podejście to jest całkowicie odwrotne niż w przypadku COM, gdzie programiści byli zmuszeni współdzielić swoje komponenty i przez to pojawiało się ryzyko, że inna aplikacja zastąpi nasze komponenty swoimi. Jeżeli korzystasz z Visual Studio .NET i dodasz odwołanie do zewnętrznego podzespołu (współdzielonego), który znajduje się w globalnym buforze podzespołów, Visual Studio nie wykona kopii lokalnej tego podzespołu. Zamiast tego bezpośrednio odwoła się do podzespołu z globalnego bufora. Działanie takie jest wymuszane przez wartość właściwości `CopyLocal`, która przy dodaniu odwołania do współdzielonego podzespołu jest ustawiana na `false`. Do właściwości tej można uzyskać dostęp, klikając podzespół prawym klawiszem myszy i wybierając z menu kontekstowego opcję *Properties*.

Podzespoły współdzielone są zaprojektowane jako rozwiązanie problemów z współdzieleniem kodu. Jednak podzespoły współdzielone muszą spełniać następujące wymagania:

- Muszą mieć globalnie unikalne nazwy, które są silne kryptograficznie.
- Muszą zawierać infrastrukturę zapobiegającą wypuszczeniu przez kogoś kolejnej wersji podzespołu i podszywaniu się w niej pod autora oryginału. Jest to realizowane za pomocą kryptografii klucza publicznego.
- Muszą zapewniać identyfikację odwołań. W czasie określania odwołania do podzespołu podzespoły współdzielone są wykorzystywane do zagwarantowania, że załadowany kod pochodzi od spodziewanego dostawcy.

Podzespoły statyczne i dynamiczne

Jak już wspominaliśmy, podzespół jest zbiorem fizycznych plików. Podzespół, który jest tworzony w czasie kompilacji i którego zależności są określane w czasie konsolidacji, jest nazywany podzespołem statycznym. Większość podzespołów, jakie będą utworzone w tej książce, jest właśnie tego typu. Dodatkowo .NET Framework zawiera zbiór klas nazywany Reflection API. Klasy te mogą być wykorzystane do tworzenia kodu na bieżąco i jego bezpośredniego wykonywania. Tego typu podzespoły są nazywane podzespołami dynamicznymi; w razie potrzeby mogą być zapisywane na dysku.

Metadane

Metadane są informacjami pozwalającymi na opisywanie komponentów. Metadane są wykorzystywane do opisywania wielu aspektów komponentów, w tym klas, metod, pól oraz podzespołów. Wspólne środowisko uruchomieniowe korzysta z metadanych do realizacji różnych zadań, na przykład kontroli poprawności podzespołu przed jego uruchomieniem czy też zbierania nieużytków w czasie działania kodu.

Czy metadane są ewolucją IDL?

Jeżeli wcześniej tworzyłeś komponenty COM, prawdopodobnie znasz podstawy języka Interface Definition Language (IDL) oraz biblioteki typów. Są one wykorzystywane do zapisywania wszystkich informacji niezbędnych dla automatyzacji COM. Można twierdzić, że metadane są podobne do IDL w świecie COM. Jednak w przeciwieństwie do IDL metadane są dokładniejsze i bardziej kompletne oraz nie są opcjonalne.

Metadane mogą być informacjami wykorzystywanymi przez wspólne środowisko uruchomieniowe do odczytywania wszystkiego na temat klas, funkcji, właściwości, zasobów i innych elementów znajdujących się w pliku wykonywalnym. Trzeba wiedzieć, że metadane są zawsze związane z plikiem zawierającym kod — oznacza to, że metadane są zawsze dołączone do samego pliku wykonywalnego lub DLL, co uniemożliwia utratę synchronizacji między kodem i metadanymi.

Wszystkie kompilatory zgodne z .NET muszą generować pełne metadane na temat każdej klasy znajdującej się w module skompilowanego kodu. W świecie COM metadane i pliki wykonywalne są zapisywane osobno. Programista COM ma możliwość zapisania biblioteki typów komponentu w postaci osobnego pliku. Dodatkowo ważne metadane COM wykorzystywane w czasie pracy, takie jak identyfikatory GUID oraz obsługiwany model wątków, są zapisywane w rejestrze. Ponieważ metadane są przechowywane osobno, a nie z komponentami COM i COM+, ich instalowanie i aktualizacja może być koszmarem. Komponent musi być zarejestrowany przed jego uruchomieniem, a jeżeli biblioteka typów jest zapisana w osobnym pliku, musi być ona zainstalowana we właściwym katalogu.

Po zainstalowaniu komponentu jego aktualizacja do nowszej wersji również może sprawiać problemy. Jeżeli zainstalujemy nowy plik binarny bez aktualizacji odpowiednich metadanych (które mogą być rozsiane po systemie), aplikacja może przestać działać. Proces instalowania i aktualizacji komponentów .NET został znacznie uproszczony. Ponieważ

wszystkie metadane skojarzone z komponentem .NET znajdują się w pliku zawierającym ten komponent, nie jest wymagana rejestracja. Po skopiowaniu nowego komponentu do systemu może być on natychmiast użyty, bez konieczności wykonywania jakichkolwiek działań konfiguracyjnych, które są wymagane w przypadku COM. Jest to nazywane instalowaniem metodą `XCOPY`.

Również aktualizacja komponentów sprawia mniej problemów, ponieważ komponenty i skojarzone z nimi metadane są zapisywane razem, dzięki czemu nie mogą się rozsynchronizować.

System wspólnego języka (CLS)

System wspólnego języka definiuje zbiór konstrukcji i ograniczeń, który służy za przewodnik dla osób piszących biblioteki i kompilatory. Języki działające w .NET Framework muszą spełniać wytyczne zapisane w CLS. Dzięki temu wszystkie języki obsługujące CLS mają pełny dostęp do bibliotek napisanych w tych językach. Można uważać CLS za podzbiór wspólnego systemu plików. Z punktu widzenia programisty, gdy jeden programista zaprojektuje publicznie dostępne klasy spełniające wymagania CLS, to można z nich korzystać w innych językach programowania. Aby utworzyć aplikację zgodną z CLS, trzeba upewnić się, że korzystamy tylko z funkcji specyficznych dla CLS w następujących częściach aplikacji:

- definicji klas publicznych;
- definicji składowych typu `public` i `protected`;
- parametrach publicznych metod klas publicznych oraz metod typu `protected`.

Na podstawie tych ograniczeń można wywnioskować, że CLS wymusza ograniczenia tylko w niektórych częściach aplikacji. Oznacza to, że pozostałe części aplikacji mogą zawierać dowolne funkcje specyficzne dla języka i nadal będzie ona zgodna z CLS.

Integracja języków na platformie .NET

Język programowania dla .NET może mieć trzy poziomy zgodności z CLS:

- **Producent** — komponenty napisane w tym języku mogą być wykorzystywane w innych językach.
- **Konsument** — w języku można korzystać z klas napisanych w innych językach programowania.
- **Rozszerzenie** — języki z tej kategorii mogą rozszerzać klasy napisane w innych językach, korzystając z funkcji dziedziczenia w .NET.

Wszystkie języki zdefiniowane w Visual Studio .NET (VB.NET, VC++ oraz C#) spełniają wszystkie trzy poziomy zgodności z CLS. Jeżeli tworzysz język przeznaczony dla wspólnego środowiska uruchomieniowego, musisz wybrać poziom zgodności pasujący do Twoich wymagań. Pełna obsługa języka w .NET Framework wymaga wiele pracy od dostawcy języka. Producent musi dostarczyć kompilator języka, który ma możliwość tworzenia kodu pośredniego zamiast kodu maszynowego. Funkcje języka muszą być zgodne

ze standardami ustalonymi przez CLS. Jeżeli przejdziesz proces zmiany architektury języka, Twój język może w zamian korzystać z takich zalet jak:

- Pełny dostęp do wszystkich zoptymalizowanych i skalowalnych bibliotek klas .NET Framework.
- Zaawansowane środowisko IDE.
- Wydajne narzędzia debugera.
- Pełna integracja języków. Przykładem tej integracji jest tworzenie interfejsu w języku COBOL i jego implementacja w C#.

Dla platformy .NET dostępnych jest kilka języków różnych firm:

- APL,
- Python,
- COBOL,
- Eiffel,
- Mercury.
- Perl,
- SmallTalk,
- Scheme,
- Oberon,

Podsumowanie

Środowisko Microsoft .NET Framework oferuje wiele nowych narzędzi do błyskawicznego tworzenia aplikacji o jakości przemysłowej. Ścisła integracja API przeznaczonych dla internetu, takich jak usługi sieciowe, XML oraz ASP.NET, powoduje, że platforma ta jest szczególnie kusząca dla aplikacji nowej generacji. Dodatkowo integracja z innymi językami daje .NET Framework przewagę nad platformami zależnymi od architektury i języka. Niezależność platformy powoduje, że .NET Framework staje się nowym liderem w tworzeniu aplikacji rozproszonych.

W tym rozdziale opisaliśmy sporo tematów. Zaczęliśmy od zadań .NET i ważnych komponentów platformy .NET. Następnie przedstawiliśmy wprowadzenie do wspólnego środowiska uruchomieniowego i omówiliśmy usługi oferowane aplikacjom przeznaczonym na platformę .NET. Omówiliśmy również trzy ważne składniki aplikacji .NET — podzespoły, moduły oraz typy. Następnie przedstawiliśmy CLS oraz zasady, jakich muszą przestrzegać niezależni dostawcy, aby utworzyć języki korzystające z usług platformy .NET.

Propozycje dalszych lektur

Witryna .NET firmy Microsoft: *http://msdn.microsoft.com/netframework*.

Duncan Mackenzie i Kent Sharkey, *Sam's Teach Yourself VB.NET in 21 Days*, Pearson Education, 2001.