Delphi. Techniki bazodanowe i internetowe

Autorzy: Peter Darakhvelidze, Eugene Markov
Tłumaczenie: Jan Ostrowski, Grzegorz Werner
ISBN: 83-7361-661-6
Tytuł oryginału: Web Services Development with Delphi
Format: B5, stron: 592

Stwórz aplikacje sieciowe, wykorzystując uniwersalne mechanizmy wymiany danych

- Wykorzystaj technologie COM, SOAP i ActiveX
- Zaimplementuj złożone mechanizmy dostępu do baz danych
- Napisz bezpieczne aplikacje, stosując szyfrowanie przesyłanych danych
- Użyj w aplikacjach języka XML

Aplikacje rozproszone są coraz częściej uruchamiane nie tylko w sieciach lokalnych, ale także w sieci WWW. Wymiana danych w tak rozległym środowisku sieciowym wymagała opracowania technologii, które zapewniałyby jednoznaczność i bezpieczeństwo przesyłanych danych. Na rynku pojawiły się mechanizmy COM i COM+, SOAP oraz kilka innych. Coraz większą popularność zyskuje język XML pozwalający na unifikację struktur danych w aplikacjach. W oparciu o takie mechanizmy powstają rozproszone aplikacje biznesowe i obliczeniowe, zwane usługami sieciowymi lub usługami WWW.

Książka „Delphi. Techniki bazodanowe i internetowe" opisuje możliwości tworzenia aplikacji sieciowych za pomocą środowiska Delphi z wykorzystaniem różnych technologii. Przedstawia kolejno najpopularniejsze z nich oraz możliwości ich zastosowania w projektach realizowanych w Delphi. Zawiera informacje poświęcone przetwarzaniu danych z wykorzystaniem języka XML oraz protokołu SOAP, a także tworzeniu usług WWW opartych na tych technologiach. W książce opisano również zagadnienia związane z korzystaniem z baz danych — systemy lokalne, mechanizmy klient-serwer oraz rozproszone, a także technologie dostępu do danych, od ADO do dbExpress.

- Mechanizmy COM i COM+
- Kontrolki ActiveX
- Architektura aplikacji bazodanowych
- Technologia DataSnap
- Aplikacje internetowe
- Obsługa gniazd w Delphi
- Kryptografia i ochrona przesyłanych danych
- Korzystanie z danych w formacie XML
- Protokół SOAP
- Stosowanie mechanizmów WebSnap

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

Spis treści

Rozdział 8.

Technologia DataSnap. Mechanizmy zdalnego dostępu

W poprzedniej części książki omówiliśmy zagadnienia związane z tworzeniem tradycyjnych aplikacji bazodanowych, które korzystają z baz danych w lokalnym komputerze albo sieci. Nie zajmowaliśmy się jednak aplikacjami, które muszą być równie dobrze przygotowane do współpracy z komputerami w sieci lokalnej oraz z wieloma maszynami zdalnymi.

Oczywiście, w takim przypadku model dostępu do danych musi być rozszerzony, ponieważ tradycyjne sposoby tworzenia aplikacji bazodanowych są nieefektywne, kiedy mamy do czynienia z dużą liczbą zdalnych komputerów.

W tym rozdziale omówimy model rozproszonej aplikacji bazodanowej, tak zwaną aplikację wielowarstwową, a ściślej — jej najprostszą wersję, trójwarstwową aplikację rozproszoną. Składa się ona z następujących części:

- serwera bazy danych,
- serwera aplikacji (oprogramowanie pośrednie),
- aplikacji klienta.

Wszystkie trzy części są połączone przez mechanizm transakcji (poziom transportu) oraz mechanizm przetwarzania danych (poziom logiki biznesowej).

Jeśli uogólnimy model trójwarstwowy, zauważymy, że zwiększanie liczby warstw nie wpływa na serwer bazy danych ani na klienta aplikacji. Dodatkowe warstwy komplikują tylko oprogramowanie pośrednie, które może na przykład zawierać serwer transakcji, serwer zabezpieczeń itp.

Wszystkie komponenty i obiekty Delphi, które umożliwiają tworzenie aplikacji wielowarstwowych określa się wspólnym mianem DataSnap.

W starszych wersjach Delphi (3, 4 i 5) komponenty te nosiły nazwę MIDAS (od ang. *Multi-tier Distributed Application Services*).

Większość komponentów omówionych w następnych rozdziałach jest dostępna na specjalnej karcie *DataSnap* palety komponentów Delphi. Do projektowania aplikacji rozproszonych potrzebnych jest jednak kilka dodatkowych komponentów, którym również poświęcimy należytą uwagę.

W tym rozdziale opiszemy następujące zagadnienia:

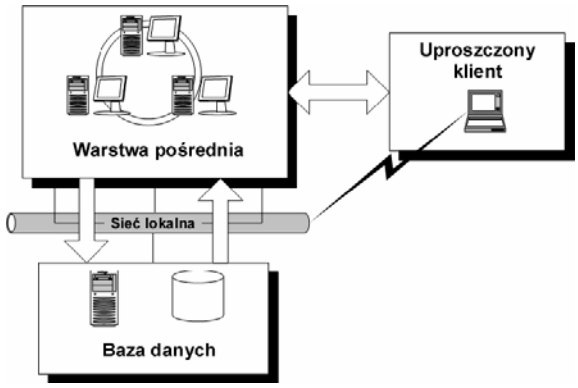
- strukturę aplikacji wielowarstwowych,
- strategię dostępu do zdalnych baz danych w technologii DataSnap,
- zdalne moduły danych,
- komponenty dostawcze,
- komponenty transakcyjne zdalnych połączeń DataSnap,
- dodatkowe komponenty — brokery połączeń.

Struktura wielowarstwowej aplikacji Delphi

Wielowarstwowa architektura aplikacji bazodanowych wzięła się z konieczności przetwarzania w serwerze żądań wielu zdalnych klientów. Na pozór zadanie to można zrealizować za pomocą tradycyjnych aplikacji typu klient-serwer, których kluczowe elementy opisaliśmy w poprzednich rozdziałach. Jeśli jednak liczba zdalnych klientów jest duża, cały ciężar przetwarzania spada na serwer bazy danych, który dysponuje dość skromnymi środkami implementowania zaawansowanej logiki biznesowej (procedury składowane, wyzwalacze, widoki itp.). Programiści są zmuszeni znacznie komplikować kod klientów, co jest wysoce niepożądane, jeśli wiele zdalnych komputerów korzysta z tego samego serwera. Kiedy oprogramowanie klienckie staje się bardziej skomplikowane, rośnie prawdopodobieństwo błędu, a świadczenie usług jest trudniejsze.

Architekturę wielowarstwową wymyślono w celu rozwiązania powyższych problemów. Wielowarstwowa aplikacja bazodanowa (zobacz rysunek 8.1) składa się z:

- „cienkich" (uproszczonych) aplikacji klienta, które zapewniają tylko usługi transmisji, prezentacji i edycji, a także podstawowego przetwarzania danych,
- jednej lub wielu warstw oprogramowania pośredniego (serwera aplikacji), które może działać w jednym komputerze albo być rozproszone w sieci lokalnej,
- serwera baz danych (Oracle, Sybase, MS SQL, InterBase itp.), który zapewnia obsługę danych i przetwarza żądania.



Rysunek 8.1. *Wielowarstwowa architektura aplikacji bazodanowej*

Zatem w tej architekturze **klienty** są bardzo prostymi aplikacjami, które zajmują się tylko transmisją danych, lokalnym buforowaniem, prezentowaniem danych za pomocą interfejsu użytkownika, edycją i podstawowym przetwarzaniem danych, stąd też często spotyka się nazwę „cienki klient" dla tego typu aplikacji.

Aplikacje klienta nigdy nie korzystają bezpośrednio z serwera bazy danych; robią to za pomocą oprogramowania pośredniego. Oprogramowanie pośrednie może składać się z jednej warstwy (w najprostszym modelu trójwarstwowym) albo mieć bardziej skomplikowaną strukturę.

Oprogramowanie pośrednie odbiera żądania klientów, przetwarza je zgodnie z zaprogramowanymi regułami logiki biznesowej, w razie potrzeby przekształca je w format odpowiedni dla serwera bazy danych, a następnie wysyła do serwera.

Serwery baz danych realizują otrzymane żądania i wysyłają wyniki do serwera aplikacji, który przekazuje je klientom.

Prostszy, trójwarstwowy model składa się z następujących elementów:

- uproszczonego klienta,
- serwera aplikacji,
- serwera bazy danych.

W naszych rozważaniach skupimy się na modelu trójwarstwowym. W środowisku programistycznym Delphi znajduje się zbiór narzędzi i komponentów do budowania klientów oraz oprogramowania pośredniego. W rozdziale 9. — „Serwer aplikacji" zajmiemy się oprogramowaniem pośrednim, a w rozdziale 10. — „Klient wielowarstwowej aplikacji rozproszonej" opiszemy kwestie projektowania klienta. Serwer aplikacji współpracuje z serwerem bazy danych, korzystając z jednej spośród technologii dostępowych obsługiwanych przez Delphi (zobacz część II — „Technologie dostępu do danych"). Są to technologie ADO, BDE, InterBase Express i dbExpress. Programista może wybrać najodpowiedniejszą technologię w zależności od konkretnego zadania i od parametrów serwera bazy danych.

Aby serwer aplikacji mógł współpracować z klientami, musi obsługiwać jeden spośród poniższych standardów rozproszonego dostępu do danych:

- Automatyzacja,
- WEB,
- CORBA,
- MTS,
- SOAP.

Aplikacje zdalnego klienta tworzy się za pomocą specjalnego zbioru komponentów, określanych wspólną nazwą DataSnap. Komponenty te zawierają standardowe mechanizmy transportu danych (DCOM, HTTP, CORBA i gniazda) i nawiązują połączenia między klientem a serwerem aplikacji. Ponadto komponenty DataSnap zapewniają klientowi dostęp do funkcji serwera aplikacji za pośrednictwem interfejsu `IAppServer` (zobacz rozdział 9. — „Serwer aplikacji").

Podczas tworzenia aplikacji klienta ważną rolę odgrywają komponenty, które zawierają zestawy danych. To również zależy od technologii dostępu do danych, co zostanie omówione w rozdziale 10. — „Klient wielowarstwowej aplikacji rozproszonej".

Oprócz wymienionych wyżej korzyści, poziom pośredni — serwer aplikacji — zapewnia dodatkowe udogodnienia, które mają duże znaczenie, jeśli chodzi o zwiększenie niezawodności i wydajności aplikacji.

Ponieważ komputery, w których działają klienty, są dość powolnymi maszynami, przeniesienie skomplikowanej logiki biznesowej na stronę serwera znacznie zwiększa ogólną wydajność rozwiązania. Nie chodzi tylko o szybszy sprzęt, ale także o optymalizację wykonywania podobnych żądań klientów. Jeśli na przykład serwer bazy danych jest przeciążony, serwer aplikacji może samodzielnie przetwarzać żądania (kolejkować je lub anulować), nie nakładając dodatkowego obciążenia na serwer bazy danych.

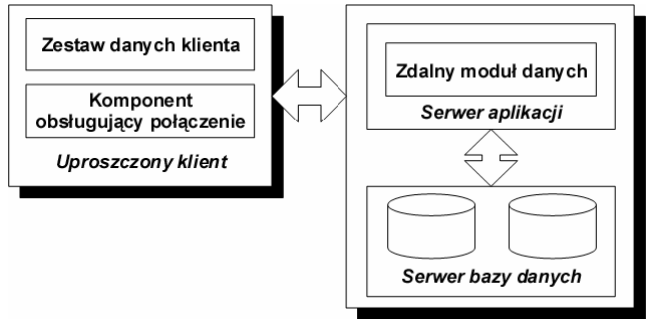
Korzystanie z serwera aplikacji zwiększa bezpieczeństwo systemu, ponieważ możesz zorganizować autoryzację użytkowników i inne środki bezpieczeństwa, nie oferując bezpośredniego dostępu do danych.

Możesz również używać chronionych kanałów komunikacji danych — na przykład protokołu HTTPS.

Trójwarstwowa aplikacja Delphi

Przyjrzyjmy się bliżej poszczególnym częściom trójwarstwowej aplikacji Delphi (zobacz rysunek 8.2). Jak już wspomniano, za pomocą Delphi można napisać zarówno część kliencką aplikacji trójwarstwowej, jak i oprogramowanie pośrednie, czyli serwer aplikacji.

Części aplikacji trójwarstwowej buduje się z komponentów DataSnap, a także innych specjalnych komponentów, które odpowiadają przede wszystkim za funkcjonowanie klienta. Dostęp do danych uzyskuje się za pomocą jednej spośród technologii obsługiwanych przez Delphi (zobacz część II — „Technologie dostępu do danych").



Rysunek 8.2. *Diagram wielowarstwowej aplikacji rozproszonej*

Podczas pisania aplikacji trójwarstwowej lepiej skorzystać ze zbioru projektów niż z pojedynczego projektu. Właśnie do tego służy narzędzie Project Manager (*View\Project Manager*).

Dane są transmitowane między serwerem aplikacji a klientami poprzez interfejs `IApp➥Server`, zapewniany przez serwer aplikacji. Interfejsu tego używają komponenty działające zarówno po stronie serwera (`TDataSetProvider`), jak i po stronie klienta (`TClientDataSet`).

Zbadajmy teraz szczegółowo poszczególne części aplikacji trójwarstwowej.

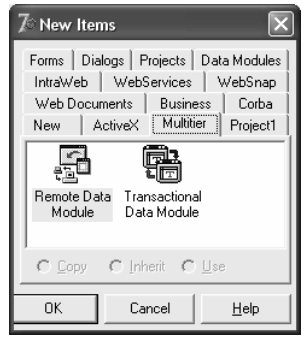
Serwery aplikacji

Serwer aplikacji realizuje logikę biznesową aplikacji rozproszonej i zapewnia klientom dostęp do bazy danych.

Z punktu widzenia programisty najważniejszą częścią każdego serwera aplikacji jest jego **zdalny moduł danych** (ang. *remote data module*). Wyjaśnijmy, dlaczego.

Po pierwsze, w zależności od implementacji, zdalny moduł danych zawiera gotowy do użycia serwer, który wystarczy tylko zarejestrować i skonfigurować. Aby utworzyć moduł danych, użyj jednej z kart *Multi-tier*, *WebSnap* lub *WebServices* magazynu Delphi (zobacz rysunek 8.3).

- *Remote Data Module* — zdalny moduł danych, który zawiera serwer Automatyzacji. Używa się go do nawiązywania połączeń za pośrednictwem DCOM, HTTP i gniazd. Więcej szczegółów znajdziesz w rozdziale 9. — „Serwer aplikacji";
- *Transactional Data Module* — zdalny moduł danych, który zawiera Microsoft Transaction Server (MTS);
- *CORBA Data Module* — zdalny moduł danych, który zawiera serwer CORBA;
- *Soap Server Data Module* — zdalny moduł danych, który zawiera serwer SOAP (ang. *Simple Object Access Protocol*);
- *WebSnap Data Module* — zdalny moduł danych, który jako serwera używa usług WWW i przeglądarki WWW.



Rysunek 8.3.
Wybieranie zdalnego modułu danych z magazynu obiektów Delphi

Po drugie, zdalny moduł danych umożliwia interakcję z klientami. Moduł udostępnia klientowi metody specjalnego interfejsu `IAppServer` (albo interfejsu wywiedzionego z `IAppServer`). Metody tego interfejsu pomagają zorganizować proces transmisji i odbierania pakietów danych z aplikacji klienta.

Po trzecie, podobnie jak tradycyjny moduł danych (zobacz rozdział 5. — „Architektura aplikacji bazodanowych"), zdalny moduł danych zawiera komponenty niewizualne oraz komponenty dostawcze. Wszystkie komponenty do obsługi połączeń i transakcji, a także komponenty, które zawierają zestawy danych zapewniają połączenie między aplikacją trójwarstwową a serwerem bazy danych. Mogą to być zbiory komponentów dla różnych technologii dostępu do danych.

Oprócz zdalnego modułu danych integralną częścią każdego serwera aplikacji są komponenty dostawcze, `TDataSetProvider`. Każdy komponent, który zawiera zestaw danych przeznaczony do przekazania klientowi musi być związany z komponentem dostawczym w zdalnym module danych.

W tym celu zdalny moduł danych musi zawierać odpowiednią liczbę komponentów `TDataSetProvider`. Komponenty te przekazują pakiety danych do klientów, a ściślej — do komponentów `TClientDataSet`. Zapewniają także dostęp do metod swojego interfejsu `IProviderSupport`. Za pomocą metod tego interfejsu można zarządzać przesyłaniem pakietów danych na niskim poziomie.

Programista zwykle nie musi tego robić. Wystarczy wiedzieć, że wszystkie komponenty obsługujące dane — zarówno po stronie klienta, jak i serwera — używają tego interfejsu. Jeśli jednak zamierzasz utworzyć własną wersję DataSnap, opis tego interfejsu będzie bardzo użyteczny (zobacz rozdział 9. — „Serwer aplikacji").

Aplikacje klienta

Aplikacja klienta w modelu trójwarstwowym powinna zawierać minimalny zbiór funkcji, a większość operacji związanych z przetwarzaniem danych powinien wykonywać serwer aplikacji.

Aplikacja zdalnego klienta musi przede wszystkim zapewnić połączenie z serwerem aplikacji. W tym celu używa się następujących komponentów DataSnap:

- `TDCOMConnection` — używa technologii DCOM,
- `TSocketConnection` — używa gniazd Windows,
- `TWebConnection` — używa protokołu HTTP,
- `TCORBAConnection` — używa połączenia w ramach architektury CORBA (niedostępny w wersji 7)
- `TSOAPConnection` — używa połączenia poprzez *WebServices*

Komponent `TSOAPConnection` zostanie omówiony oddzielnie.

Komponenty połączeniowe DataSnap używają interfejsu `IAppServer`, który korzysta z komponentów dostawczych działających po stronie serwera, oraz klienckich komponentów `TClientDataSet` do przekazywania pakietów danych.

Dane są obsługiwane przez komponenty `TClientDataSet`, które działają w trybie buforowania danych.

Do prezentacji danych i tworzenia interfejsu użytkownika w aplikacji klienta służą standardowe formanty z karty *Data Controls* palety komponentów Delphi.

Więcej informacji o projektowaniu klientów w wielowarstwowych aplikacjach bazodanowych znajdziesz w rozdziale 10. — „Klient wielowarstwowej aplikacji rozproszonej”.

Mechanizm zdalnego dostępu DataSnap

Do przekazywania danych między komponentem dostawczym a klienckim zestawem danych (zobacz rysunek 8.2) niezbędne jest utworzenie łącza transportowego między klientem a serwerem, które zapewnia fizyczną transmisję danych. Można to zrobić za pomocą różnych protokołów transportowych obsługiwanych przez system operacyjny. Różne typy połączeń, które umożliwiają skonfigurowanie kanału komunikacyjnego i rozpoczęcie wysyłania i odbierania informacji, są zawarte w kilku komponentach DataSnap. Aby utworzyć po stronie klienta połączenie korzystające z konkretnego protokołu transportowego, wystarczy umieścić na formularzu odpowiedni komponent i prawidłowo ustawić kilka jego właściwości. Komponent ten będzie współpracował ze zdalnym modułem danych, który stanowi część serwera aplikacji.

Poniżej opiszemy komponenty połączeniowe, które obsługują takie protokoły jak DCOM, gniazda TCP/IP, HTTP i CORBA.

Komponent TDCOMConnection

Komponent `TDCOMConnection` obsługuje transmisję danych z wykorzystaniem technologii Distributed COM lub COM+ i służy przede wszystkim do nawiązywania połączeń w obrębie sieci lokalnej.

Aby skonfigurować połączenie DCOM, musisz najpierw podać nazwę komputera, w którym zainstalowany jest serwer aplikacji. W przypadku komponentów `TDCOMConn➥ection` musi to być zarejestrowany serwer Automatyzacji. Nazwę komputera określa właściwość:

```
property ComputerName: String;
```

Jeśli nazwa jest prawidłowa, właściwość

```
property ServerName: String;
```

umożliwia wybór jednego spośród dostępnych serwerów w oknie *Object Inspector*.

Po wybraniu serwera właściwość

```
property ServerGUID: String;
```

jest wypełniana automatycznie globalnym identyfikatorem zarejestrowanego serwera Automatyzacji.

Aby klient mógł pomyślnie połączyć się z serwerem aplikacji, trzeba ustawić obie właściwości w opisanej wyżej kolejności. Sama nazwa serwera albo sam identyfikator GUID nie zagwarantuje dostępu do zdalnego obiektu COM.

Do otwierania i zamykania połączenia służy właściwość

```
property Connected: Boolean;
```

albo metody

```
procedure Open;
procedure Close;
```

Transmisją danych między klientem a serwerem zarządza interfejs `IAppServer` komponentu `TDCOMConnection`:

```
property AppServer: Variant;
```

Dostęp do tego interfejsu można uzyskać również za pomocą metody:

```
function GetServer: IAppServer; override;
```

Właściwość

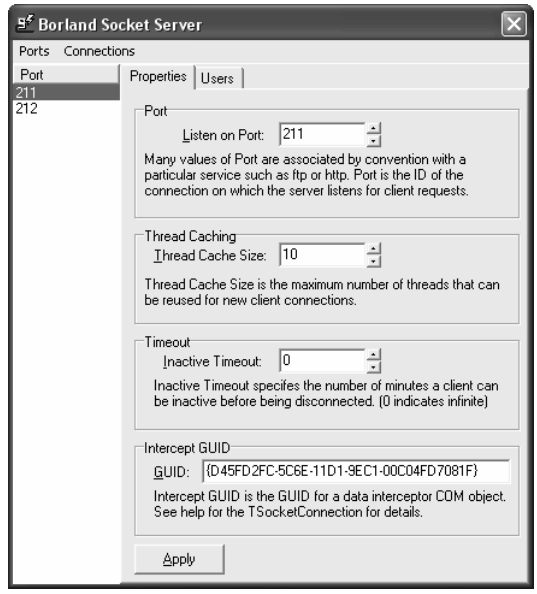
```
property ObjectBroker: TCustomObjectBroker;
```

pozwala skorzystać z instancji komponentu `TSimpleObjectBroker`, aby uzyskać listę dostępnych serwerów w czasie wykonywania aplikacji.

Metody obsługi zdarzeń komponentu `TDCOMConnection` są wymienione w tabeli 8.1.

Komponent TSocketConnection

Komponent `TSocketConnection` zapewnia połączenie między klientem a serwerem z wykorzystaniem gniazd TCP/IP. Aby połączenie zostało pomyślnie otwarte, serwerowa część aplikacji musi być wyposażona w serwer gniazd (aplikacja *ScktSrvr.exe*; zobacz rysunek 8.4).



Rysunek 8.4. *Serwer gniazd ScktSrvr.exe*

Tabela 8.1. *Metody obsługi zdarzeń komponentu TDCOMConnection*

Deklaracja	Opis
`property AfterConnect: TNotifyEvent;`	Wywoływana po nawiązaniu połączenia.
`property AfterDisconnect: TNotifyEvent;`	Wywoływana po zamknięciu połączenia.
`property BeforeConnect: TNotifyEvent;`	Wywoływana przed nawiązaniem połączenia.
`property BeforeDisconnect: TNotifyEvent;`	Wywoływana przed zamknięciem połączenia.
`type TGetUsernameEvent = procedure(Sender: TObject); var Username: String) of object;` `property OnGetUserName: TGetUsernameEvent;`	Wywoływana tuż przed wyświetleniem okna logowania, które służy do autoryzacji zdalnego użytkownika. Dzieje się tak, kiedy właściwość `LoginPrompt` jest ustawiona na `True`. Parametr `Username` może zawierać domyślną nazwę użytkownika, która pojawi się w oknie logowania.
`type TLoginEvent = procedure(Sender: TObject; Username, Password: String) of object;` `property OnLogin: TLoginEvent;`	Wywoływana po nawiązaniu połączenia, jeśli właściwość `LoginPrompt` jest ustawiona na `True`. Parametry `Username` i `Password` zawierają nazwę użytkownika i hasło wprowadzone podczas autoryzacji.

Właściwość

```
property Host: String;
```

musi zawierać nazwę komputera z serwerem. Ponadto właściwość

```
property Address: String;
```

musi zawierać adres serwera.

W celu otwarcia połączenia trzeba ustawić obie powyższe właściwości.

Właściwość

```
property Port: Integer;
```

określa numer portu. Domyślny numer portu to 211, ale programista może zmienić port, na przykład po to, aby umożliwić używanie go przez różne kategorie użytkowników albo utworzyć chroniony kanał komunikacji.

Jeśli nazwa komputera została określona prawidłowo, obok właściwości

```
property ServerName: String;
```

w oknie *Object Inspector* pojawi się lista dostępnych serwerów Automatyzacji. Po wybraniu serwera zostanie automatycznie ustawiona właściwość

```
property ServerGUID: String;
```

która zawiera identyfikator GUID zarejestrowanego serwera. Można również ustawić ją ręcznie.

Metoda

```
function GetServerList: OleVariant; virtual;
```

zwraca listę zarejestrowanych serwerów Automatyzacji.

Do otwierania i zamykania połączenia służy właściwość

```
property Connected: Boolean;
```

albo jedna z metod

```
procedure Open;
procedure Close;
```

Kanał gniazda TCP/IP można szyfrować za pomocą właściwości

```
property InterceptName: String;
```

która zawiera identyfikator obiektu COM obsługującego szyfrowanie-deszyfrowanie kanału. Identyfikator GUID tego obiektu jest określony przez właściwość:

```
property InterceptGUID: String;
```

Wskazany obiekt COM przechwytuje dane przechodzące przez kanał i przetwarza je zgodnie z własnym kodem. Może to być szyfrowanie, kompresja, manipulowanie szumem itp.

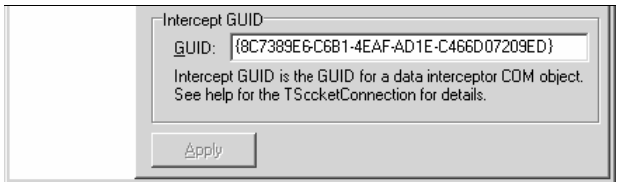
Obiekt COM, który zapewnia dodatkowe przetwarzanie danych przechodzących przez kanał, musi zostać utworzony przez programistę. Obiekt przechwytujący dane musi obsługiwać standardowy interfejs `IDataIntercept`.

Oczywiście, po stronie serwera musi znajdować się zarejestrowany obiekt COM, który wykonuje odwrotną operację. W tym celu używa się serwera gniazd (zobacz rysunek 8.5). Łańcuch *Intercept GUID* musi zawierać identyfikator GUID obiektu przechwytującego dane.

Metoda

```
function GetInterceptorList: OleVariant; virtual;
```

zwraca listę obiektów przechwytujących zarejestrowanych w serwerze.



Rysunek 8.5. *Rejestrowanie przechwytującego obiektu COM w serwerze gniazd*

Komponent `TSocketConnection` udostępnia interfejs `IAppServer`, który organizuje proces transmisji danych między klientem a serwerem:

```
property AppServer: Variant;
```

Dostęp do tego interfejsu można również uzyskać za pomocą metody:

```
function GetServer: IAppServer; override;
```

Właściwość

```
property ObjectBroker: TCustomObjectBroker;
```

pozwala skorzystać z instancji komponentu `TSimpleObjectBroker`, aby uzyskać listę dostępnych serwerów w czasie wykonywania aplikacji.

Metody obsługi zdarzeń komponentu `TSocketConnection` są takie same jak metody komponentu `TDCOMConnection` (zobacz tabela 8.1).

Komponent TWebConnection

Komponent `TWebConnection` łączy klienta z serwerem za pomocą protokołu transportowego HTTP. W celu zarządzania tym komponentem trzeba zarejestrować bibliotekę *wininet.dll* w komputerze z klientem. Zwykle nie wymaga to dodatkowej pracy, ponieważ plik ten znajduje się w systemowym folderze Windows, jeśli w komputerze zainstalowany jest Internet Explorer.

W komputerze z serwerem musi być zainstalowane oprogramowanie Internet Information Server 4.0 (lub nowsza wersja) albo Netscape Enterprise 3.6 (lub nowsza wersja). Oprogramowanie to zapewnia komponentowi `TWebConnection` dostęp do biblioteki dynamicznej *HTTPsrvr.dll* (znajdziesz tę bibliotekę w katalogu BIN Delphi), która również powinna być umieszczona w serwerze. Jeśli na przykład plik *HTTPsrvr.dll* jest umieszczony w folderze *Scripts* programu IIS 4.0 w serwerze *www.someserver.com*, to właściwość

```
property URL: String;
```

powinna zawierać następującą wartość:

```
http://someserver.com/scripts/httpsrvr.dll
```

Jeśli adres URL jest prawidłowy i serwer jest odpowiednio skonfigurowany, to przy właściwości

```
property ServerName: String;
```

w oknie *Object Inspector* pojawi się lista wszystkich nazw zarejestrowanych serwerów aplikacji. Jedną z tych nazw trzeba przypisać właściwości `ServerName`.

Po ustawieniu nazwy serwera jego identyfikator GUID zostanie automatycznie przypisany właściwości:

```
property ServerGUID: String;
```

Właściwości

```
property UserName: String;
```

oraz

```
property Password: String;
```

w razie potrzeby mogą zawierać nazwę i hasło użytkownika, które zostaną użyte podczas autoryzacji.

Właściwość

```
property Proxy: String;
```

zawiera nazwę serwera pośredniczącego (proxy).

Możesz dołączyć nazwę aplikacji do nagłówka komunikatu HTTP, korzystając z właściwości:

```
property Agent: String;
```

Do otwierania i zamykania połączenia służy właściwość:

```
property Connected: Boolean;
```

Podobne funkcje pełnią metody:

```
procedure Open;
procedure Close;
```

Dostęp do interfejsu `IAppServer` można uzyskać za pomocą właściwości

```
property AppServer: Variant;
```

albo metody

```
function GetServer: IAppServer; override;
```

Listę dostępnych serwerów aplikacji zwraca metoda:

```
function GetServerList: OleVariant; virtual;
```

Właściwość

```
property ObjectBroker: TCustomObjectBroker;
```

pozwala skorzystać z instancji komponentu `TSimpleObjectBroker`, aby uzyskać listę dostępnych serwerów w czasie wykonywania aplikacji.

Metody obsługi zdarzeń komponentu `TWebConnection` są takie same jak metody komponentu `TDCOMConnection` (zobacz tabela 8.1).

Komponent TCORBAConnection

Komponent `TCORBAConnection` zapewnia aplikacji klienta dostęp do serwera CORBA. W celu skonfigurowania połączenia z serwerem wystarczy ustawić jedną właściwość:

```
type TRepositoryId = type string;
property RepositoryId: TRepositoryId;
```

Musimy podać nazwę serwera i zdalnego modułu danych, oddzielając je ukośnikiem. Jeśli na przykład nazwa serwera to `CORBAServer`, a nazwa modułu to `CORBAModule`, właściwość ta powinna mieć wartość:

```
CORBAServer/CORBAModule
```

Adres serwera można również podać w notacji IDL (od ang. *Interface Definition Language*):

```
IDL: CORBAServer/CORBAModule:1.0
```

Właściwość

```
property HostName: String;
```

powinna zawierać nazwę komputera z serwerem albo jego adres IP. Jeśli nie ustawisz tej właściwości, komponent `TCORBAConnection` połączy się z pierwszym znalezionym serwerem, który ma parametry określone przez właściwość `RepositoryId`. Nazwa serwera CORBA jest zawarta we właściwości:

```
property ObjectName: String;
```

Do otwierania i zamykania połączenia służy właściwość:

```
property Connected: Boolean;
```

Podobne funkcje pełnią metody:

```
procedure Open;
procedure Close;
```

Jeśli z jakiegoś powodu połączenie nie zostanie otwarte, można zapobiec zawieszeniu aplikacji za pomocą poniższej właściwości:

```
property Cancelable: Boolean;
```

Kiedy ta właściwość jest ustawiona na `True`, połączenie zostanie anulowane, jeśli nie uda się go nawiązać w czasie jednej sekundy. Możesz również ustawić metodę obsługi zdarzenia:

```
type TCancelEvent = procedure(Sender: TObject; var Cancel: Boolean;
var DialogMessage: String) of object;
property OnCancel: TCancelEvent;
```

która jest wywoływana przed anulowaniem połączenia.

Kiedy połączenie zostanie otwarte, klient może uzyskać dostęp do interfejsu `IAppServer` za pomocą właściwości

```
property AppServer: Variant;
```

albo metody

```
function GetServer: IAppServer; override;
```

Metody obsługi zdarzenia komponentu `TCORBAConnection` są dziedziczone (z wyjątkiem opisanej wyżej metody `Cancel`) po klasie `TCustomConnection`. Metody te opisano w tabeli 8.1.

TCORBAConnection nie jest dostępny w Delphi 7. Korzystanie z technologii CORBA jest możliwe w Delphi 7, ale nie w połączeniu z technologią DataSnap.

Dodatkowe komponenty — brokery połączeń

Kolekcja DataSnap zawiera zestaw dodatkowych komponentów, które ułatwiają zarządzanie połączeniami między zdalnymi klientami a serwerami aplikacji. Wyjaśnijmy, jak się ich używa.

Komponent TSimpleObjectBroker

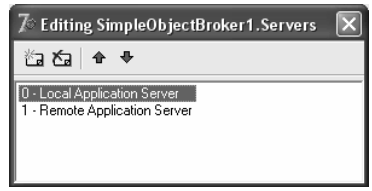
Komponent `TSimpleObjectBroker` zawiera listę serwerów, z których mogą korzystać klienty wielowarstwowej aplikacji rozproszonej. Listę tę tworzy się w fazie projektowania aplikacji. W razie potrzeby (na przykład jeśli serwer będzie nieczynny lub przeciążony), komponent połączeniowy aplikacji klienta może użyć jednego spośród dodatkowych serwerów z listy komponentu `TSimpleObjectBroker` w czasie działania aplikacji.

Aby włączyć tę funkcję, podaj listę serwerów w komponencie `TSimpleObjectBroker` i zdefiniuj wskaźnik do niego we właściwości `ObjectBroker` swojego komponentu połączeniowego (zobacz wyżej). Jeśli połączenie zostanie ponownie nawiązane, nazwę serwera będzie można pobrać z listy komponentu `TSimpleObjectBroker`.

Lista serwerów jest zdefiniowana przez właściwość:

```
property Server: TServerCollection;
```

W fazie projektowania aplikacji listę serwerów tworzy się za pomocą specjalnego edytora (zobacz rysunek 8.6), który można wywołać przez kliknięcie przycisku właściwości w oknie *Object Inspector*.



Rysunek 8.6. *Edytor listy serwerów przechowywanej w komponencie TSimpleObjectBroker*

Właściwość `Servers` to kolekcja obiektów `TServerItem`. Klasa ta ma kilka właściwości, które umożliwiają opisanie kluczowych parametrów serwera.

Właściwość

```
property ComputerName: String;
```

definiuje nazwę komputera, w którym działa serwer aplikacji. Możesz też ustawić nazwę serwera, która będzie wyświetlana na liście serwerów:

```
property DisplayName: String;
```

Za pomocą poniższej właściwości możesz określić, czy rekord serwera będzie dostępny, czy niedostępny:

```
property Enabled: Boolean;
```

Kiedy próba użycia rekordu z listy nie powiedzie się, właściwość

```
property HasFailed: Boolean;
```

przybierze wartość `True` i od tego momentu rekord będzie ignorowany.

Właściwość

```
property Port: Integer;
```

zawiera numer portu, którego użyto w celu połączenia się z serwerem.

Kiedy połączenie jest nawiązywane, wartości powyższych właściwości są podstawiane w miejsce odpowiednich właściwości komponentu połączeniowego:

```
DCOMConnection.ComputerName :=
TSimpleObjectBroker(DCOMConnection.ObjectBroker).Servers[0].ComputerName;
```

Oprócz listy serwerów komponent `TSimpleObjectBroker` zawiera tylko kilka dodatkowych właściwości i metod.

Metoda

```
function GetComputerForGUID(GUID: TGUID): String; override;
```

zwraca nazwę komputera, w którym zarejestrowany jest serwer z identyfikatorem GUID określonym przez parametr `GUID`.

Metoda

```
function GetComputerForProgID(const ProgID): String; override;
```

zwraca nazwę komputera, w którym zarejestrowany jest serwer o nazwie określonej przez parametr `ProgID`.

Właściwość

```
property LoadBalance: Boolean;
```

służy do wybierania serwera z listy. Jeśli ma wartość `True`, serwer jest wybierany losowo; w przeciwnym razie sugerowana jest pierwsza dostępna nazwa serwera.

Komponent TLocalConnection

Komponentu TLocalConnection używa się lokalnie w celu uzyskania dostępu do istniejących komponentów dostawczych.

Właściwość

```
property Providers[const ProviderName: String]: TCustomerProvider;
```

zawiera wskaźniki do wszystkich komponentów dostawczych, które znajdują się w tym samym module co dany komponent TLocalConnection. Poszczególne pozycje tej listy są indeksowane według nazw komponentów dostawczych.

Łączną liczbę komponentów dostawczych na liście zwraca właściwość:

```
property ProviderCount: Integer;
```

Ponadto, stosując komponent TLocalConnection, możemy uzyskać lokalny dostęp do interfejsu IAppServer. Użyj właściwości

```
property AppServer: IAppServer;
```

albo metody

```
function GetServer: IAppServer; override;
```

Komponent TSharedConnection

Jeśli interfejs IAppServer zdalnego modułu danych używa metody, która zwraca wskaźnik do analogicznego interfejsu innego zdalnego modułu danych, to pierwszy moduł nazywamy macierzystym, a drugi — potomnym (zobacz rozdział 9. — „Serwer aplikacji"). Komponent TSharedConnection łączy aplikację klienta z potomnym modułem danych serwera aplikacji.

Właściwość

```
property ParentConnection: TDispatchConnection;
```

powinna zawierać wskaźnik do komponentu połączeniowego z macierzystym zdalnym modułem danych serwera aplikacji. Nazwę modułu potomnego określa właściwość:

```
property ChildName: String;
```

Jeśli interfejs macierzystego zdalnego modułu danych został skonfigurowany prawidłowo, lista w oknie *Object Inspector* będzie pokazywać nazwy wszystkich modułów potomnych.

Interfejs IAppServer potomnego zdalnego modułu danych jest zwracany przez właściwość

```
property AppServer: Variant;
```

albo metodę

```
function GetServer: IAppServer; override;
```

Metody obsługi zdarzenia komponentu TSharedConnection są dziedziczone po klasie TCustomConnection. (zobacz tabela 8.1).

Komponent TConnectionBroker

Komponent `TConnectionBroker` zapewnia scentralizowaną kontrolę nad połączeniem między klienckimi zestawami danych a serwerem aplikacji. Właściwość `Connection➥Broker` we wszystkich klienckich zestawach danych musi wskazywać instancję komponentu `TConnectionBroker`. Dzięki temu zmiana rodzaju połączenia (na przykład z HTTP na gniazda TCP/IP) nie wymaga modyfikowania właściwości `RemoteServer` we wszystkich komponentach `TClientDataSet` — wystarczy zmienić właściwość:

```
property Connection: TCustomRemoteServer;
```

Dostęp do interfejsu `IAppServer` można uzyskać za pomocą właściwości

```
property AppServer: Variant;
```

albo metody

```
function GetServer: IAppServer; override;
```

Metody obsługi zdarzeń komponentu `TConnectionBroker` opisano w tabeli 8.1.

Podsumowanie

Wielowarstwowe aplikacje rozproszone zapewniają efektywną interakcję między wieloma zdalnymi, uproszczonymi klientami a serwerami aplikacji za pomocą oprogramowania pośredniego. W aplikacjach wielowarstwowych najczęściej używa się modelu trójwarstwowego, w którym oprogramowanie pośrednie składa się tylko z serwera aplikacji.

W Delphi do tworzenia aplikacji trójwarstwowych używa się komponentów DataSnap i zdalnych modułów danych. Wszystkie te narzędzia są dostępne w wersjach obsługujących różne protokoły transportowe.

Trójwarstwowe aplikacje rozproszone korzystają też z komponentów `TDataSetProvider` oraz `TClientDataSet`, które obsługują zestawy danych po stronie klienta.