STUDIA INFORMATICA 2000
Volume 21 Number 1 (39)

Henryk KRAWCZYK, Maciej LIPCZYŃSKI
Politechnika Gdańska, Katedra Architektury Systemów Komputerowych

PROJEKTOWANIE WIELOWARSTWOWEJ ARCHITEKTURY APLIKACJI SIECIOWYCH I INTERNETOWYCH[1]

Streszczenie. Zaprezentowano i scharakteryzowano główne warstwy aplikacji sieciowych i internetowych. Przedstawiono problemy projektowania i implementacji struktury trójwarstwowej i jej wykorzystania do budowy internetowego systemu wspierającego diagnostykę i rozpoznawanie zdjęć endoskopowych.

DESIGN OF MULTI-LAYER ARCHITECTURE FOR NETWORKED AND INTERNET APPLICATIONS

Summary. Main layers of networked and Internet applications are presented and characterised. Programming and implementation problems of 3-tier architecture are discussed and its suitability for design of a medical system to support diagnoses based on endoscope image recognition.

1. Wprowadzenie

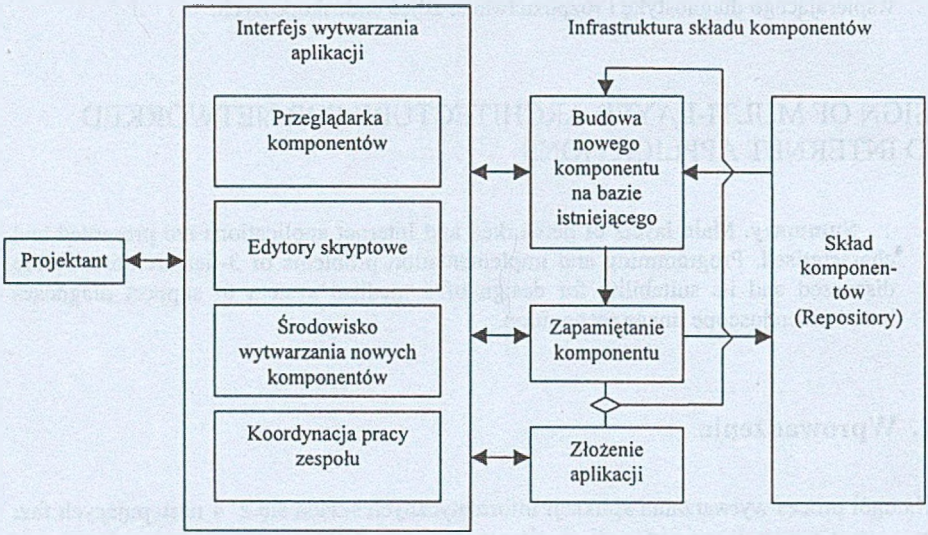
Na ogół proces wytwarzania aplikacji informatycznych składa się z 4 następujących faz: analiza, projektowanie, implementacja i utrzymanie [8]. Bardzo krótko można je scharakteryzować w następujący sposób:

- analiza – gdzie precyzuje się problem i definiuje się ogólny model aplikacji,
- projektowanie – gdzie prezentuje się szczegółowe rozwiązanie problemu w formie podstawowych komponentów aplikacji,

[1] Pracę wykonano w ramach grantu KBN Nr 8T11C 00117 „Komputerowa archiwizacja i wspomaganie rozpoznawania zdjęć gastroenterologicznych na podstawie równoległych algorytmów klasyfikujących"

- implementacja – gdzie wytworzoną aplikację wyraża się w danym języku programowania,
- utrzymanie – gdzie dokonuje się modyfikacji aplikacji w celu spełnienia zmieniających się wymagań.

Rozwój sieci komputerowych, potrzeba budowy nowego rodzaju aplikacji, tzw. aplikacji rozproszonych czy zespolonych, sprawia, że zmienia się charakter wyżej wymienionych faz. Obecnie odchodzi się od metodologii zorientowanych na bezpośrednie otrzymywanie kodu źródłowego aplikacji na rzecz określenia odpowiednich komponentów i ich integracji we właściwą aplikację (component-based applications [7]). Dzięki temu ukrywa się przed projektantem wiele szczegółów technicznych i ułatwia się mu realizację całego procesu wytwarzania. Środowisko zapewniające taki sposób budowy aplikacji – IDE (Integrated Development Environment) przedstawia rys. 1. Składa się ono z dwóch zasadniczych części złożonego interfejsu (4 zasadniczych modułów) i infrastruktury składu (ang. repository), umożliwiającego modyfikację i integrację komponentów.



Rys. 1. Zintegrowane środowisko wytwarzania aplikacji (IDE) złożonej z komponentów
Fig. 1. Integrated development environment for Component-based applications

Implementacja poszczególnych komponentów aplikacji bazuje na tzw. platformie DCP (Distributed Component Platform [4]), która w pełni definiuje każdy komponent aplikacji, jego zewnętrzny interfejs i protokoły interakcji z innymi komponentami. Do opisu komponentów wykorzystuje się podejście obiektowe, gdzie własności i metody określają odpowiednie API (Application Programming Interface), zaś pojawiające się zdarzenie charakteryzuje odpowiedź aplikacji na zewnętrzne pobudzenie lub zmianę jej warunków

wewnętrznych. Przykładem platformy DCP jest DCOM (Distributed Component Object Model [7]) – firmy Microsoft oraz JavaBeans –firmy Sun [7]. Platforma DCOM jest niezależna od języka programowania, ale zależna od środowiska obliczeniowego (Windows), zaś JavaBeans – odwrotnie, jest niezależna od środowiska i zależna od języka programowania (Java).

Grupa OMG (Object Management Group), która poprzednio opracowała otwarty system specyfikacji dla rozproszonych obiektów obliczeniowych, sprecyzowała ostatnio standard dla komponentów aplikacji. W ten sposób określono nową architekturę CORBA (Common Object Request Broker Architecture [9]), obejmującą opis komponentów dla realizacji różnych usług oraz model programowania interfejsu - IDL (Interface Definition Language).

Te dwa powyższe podejścia: DCOM i CORBA nie uwzględniają specyfiki aplikacji WWW wykorzystujących język opisu dokumentów HTML [5]. Innymi słowy, obecne implementacje tego typu aplikacji nie odpowiadają abstrakcjom reprezentowanym w podejściach obiektowych. Niemniej czynione są próby integracji podejścia obiektowego i skryptowego, czego przykładem jest WCML (Web Composition Model Language [7]). Oprócz tego dąży się, by obiekty klienta współpracowały z obiektami serwera za pomocą protokołów opracowanych przez grupę OMG, tj. Internet Inter ORB Protocol lub Java's Remote Method Invocation. Podobnie DOM (W3C's Document Object Model) umożliwia aplikacjom dostęp do danych WWW poprzez definicję obiektowego API (dla dokumentów HTML i XML).

W pracy skoncentrowano się na sposobie wytwarzania tzw. trójwarstwowych architektur (3-tier architecture) dla zastosowań medycznych. W tym celu przeanalizowano podstawowe platformy programistyczne dla komponentów: Java [3], CORBA [9] oraz Java + CORBA [2]. Na tej podstawie przedstawiono metodę projektowania i implementacji aplikacji WWW oraz przytoczono trójwarstwowy model pracującego w Internecie systemu medycznego wspomagającego pracę gastrologa.

2. Platformy przetwarzania rozproszonego

Poniżej przedstawiono wybrane platformy przetwarzania rozproszonego, które mają zastosowanie we wspomnianym wyżej systemie wspomagającym pracę gastrologa.

2.1. Platforma JAVA

Java to platforma przetwarzania, jak i język programowania wysokiego poziomu, który charakteryzuje się bardzo cennymi, z punktu widzenia aplikacji, cechami, takimi jak:

- obiektowość, wielowątkowość, dynamiczność,
- wydajność, stabilność, bezpieczeństwo,
- neutralność w stosunku do architektury sprzętu, przenośność, interpretowalność,
- prostota, wysoka jakość,.

Najważniejszymi zaletami języka Java jest obiektowość [8] i niezależność od platformy systemowej. Raz napisany kod może być bez przeszkód uruchamiany na komputerach o rozmaitej konfiguracji, działających pod kontrolą różnych systemów operacyjnych. Stało się to możliwe dzięki zastosowaniu, jako wyniku kompilacji plików źródłowych, tzw. kodu bajtowego Javy. Do uruchomienia programu w Javie potrzebny jest interpreter działający pod kontrolą odpowiedniego systemu operacyjnego i tylko on jest zależny od platformy systemowej. Wynikiem działania interpretera jest kod w postaci akceptowanej przez dany system operacyjny i możliwy do wielokrotnego wykonania w danym środowisku obliczeniowym. Taka możliwość jest odzwierciedleniem hasła promującego najnowszą wersję języka Java (wersja 1.3): „wystarczy raz napisać kod, a program można uruchomić wszędzie".

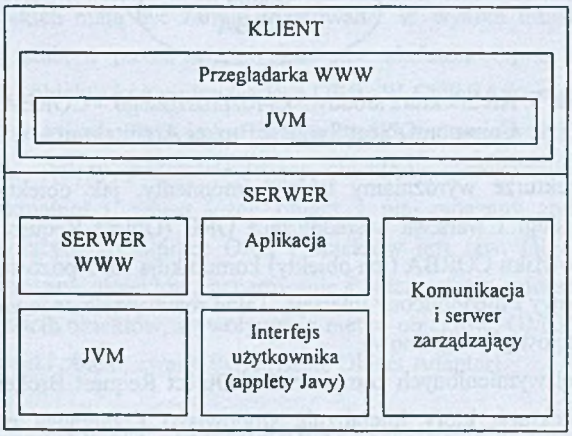
Platforma Javy różni się od większości innych tym, że jest to środowisko wyłącznie programowe – działa ono na innych, sprzętowo zależnych platformach. Większość takich platform jest opisywana jako kombinacja sprzętu i systemu operacyjnego, natomiast platforma Javy składa się z dwóch komponentów: maszyny wirtualnej i interfejsu API. Każdy interpreter kodu bajtowego stanowi samodzielny program czy układ elektroniczny, lub np. komponent przeglądarek internetowych. Java API jest zbiorem bardzo wielu gotowych do wykorzystania bibliotek komponentów programowych. Są one pogrupowane w tzw. pakiety. Maszyna wirtualna i interfejs API izolują każdy program napisany w Javie od sprzętu, co uniezależnia go od specyfiki wykorzystanego środowiska obliczeniowego.

Rosnące zainteresowanie tak bogatym językiem, a także strategia producenta polegająca na promowaniu Javy jako idealnej platformy sieciowej zaowocowały wysoką jakością najnowszej wersji, która jest bardziej funkcjonalna, bezpieczniejsza, szybsza w porównaniu z poprzednimi wersjami. Jest tez z nimi kompatybilna. Programy napisane w tym nowym języku są jednak bardzo wolne, zwłaszcza jeśli posiadają interfejs graficzny. Szacuje się, że są one nawet 20-krotnie wolniejsze od aplikacji napisanych w innych popularnych językach np. C++!

W najnowszej wersji języka Java (wersja 1.2.2 lub 1.3), która jest przeznaczona między innymi do tworzenia aplikacji użytkowych dla przedsiębiorstw, wprowadzono dużo innowacji. Duży nacisk został położony m.in. na zwiększenie poufności przetwarzanych danych. Został zastosowany zupełnie nowy system zabezpieczeń o nazwie Java Security Model. Cechuje się on głównie możliwością dostosowania poziomu zabezpieczeń do

specyficznych wymagań zależnych od potrzeb. Dostęp do zasobów i plików jest teraz lepiej kontrolowany. Możliwe jest także szyfrowanie danych poprzez zarządzanie certyfikatami, prywatnymi i publicznymi kluczami kryptograficznymi, a także cyfrowymi podpisami. Na uwagę zasługuje również ulepszony mechanizm zarządzania pamięcią.

Część swych zalet Java może wykazać dzięki powszechnemu stosowaniu w Internecie języka HTML. Dzięki odpowiednim znacznikom jest możliwe znalezienie, załadowanie i uruchomienie prostych programów (appletów) po stronie klienta. Taki łatwy i elastyczny sposób oferowania programów użytkownikom możliwy jest wtedy, kiedy są im one potrzebne. Można więc powiedzieć, że WWW i Java upraszczają zadanie wytwarzania klienckich komponentów aplikacji internetowych. Schemat blokowy modułu klient – serwer przedstawiony jest na rys. 2. Klient, reprezentujący żądanie użytkownika, wymaga tylko przeglądarki. Cały ciężar utrzymania aplikacji spada na projektanta aplikacji (serwera). Praca serwera wymaga funkcjonowania pięciu modułów. Odpowiadają one za wykonanie, komunikację oraz zarządzanie żądaniami, które mogą przychodzić od wielu użytkowników.

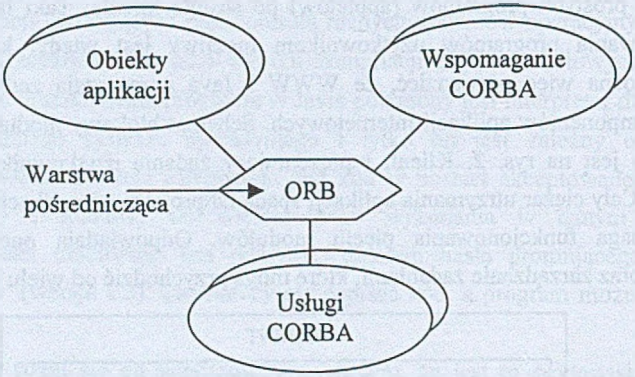


Rys. 2. Architektura modelu klient – serwer w Javie
Fig. 2. Java Architecture of client – server model

Wszystkie zmiany wprowadzone w najnowszych wersjach języka Java spowodowały, że stał się on nowoczesną platformą, dzięki której można pisać nie tylko proste programy, ale również budować funkcjonalne, bezpieczne, poważne aplikacje dla różnych przedsiębiorstw i instytucji. Nie jest to już tylko język służący do tworzenia prostych appletów. Opracowywane są bardzo nowoczesne technologie (np. Jini - [10]), które zmienią zupełnie sposób funkcjonowania urządzeń elektronicznych oraz komunikacji między nimi. Java, wykorzystując te technologie, służy jako pomost pozwalający na współpracę tych urządzeń i staje się naprawdę uniwersalnym językiem programowania.

2.2. Środowisko CORBA

Standaryzacja oprogramowania systemów rozproszonych przez OMG (Object Management Group) przyczyniła się do definicji nowego środowiska CORBA (Common Object Request Broker Architecture), którego zadaniem jest między innymi ustalenie wspólnego interfejsu dla aplikacji rozproszonych. Schemat tej architektury przedstawia rys. 3.



Rys. 3. Architektura środowiska rozproszonego – CORBA
Fig. 3. Common Object Request Broker Architekture – CORBA

W tej architekturze wyróżniamy takie komponenty, jak obiekty aplikacji, obiekty wspomagające, usługi i warstwa pośrednicząca ORB (Object Request Broker). Aplikacje pracujące w środowisku CORBA (ich obiekty) komunikują się z pozostałymi komponentami poprzez standardowy i ujednolicony interfejs. Takie rozwiązanie zapewnia elastyczność oraz niezależność sprzętową i programową.

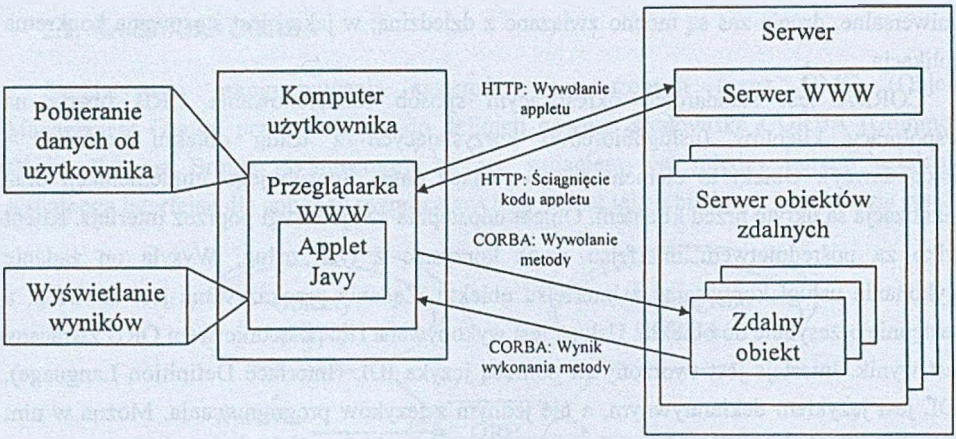
Spośród wyżej wymienionych komponentów Object Request Broker jest podstawowym elementem architektury, który inicjalizuje środowisko i zarządza całością komunikacji pomiędzy jego elementami. Umożliwia także obiektom dokonywanie interakcji w heterogenicznym, rozproszonym środowisku, niezależnym od platform, na których istnieją te obiekty, ani od technik, jakie zostały użyte do ich implementacji. ORB, wykonując swoje zadania, bazuje na obiektach usługowych, które są ogólnie odpowiedzialne za zarządzanie i obsługiwanie obiektów aplikacji, np. tworzenie obiektów, kontrola dostępu, nadzorowanie działania relokowanych obiektów itp. Obiekty wspomagające są wykorzystywane przy tworzeniu i rozwijaniu aplikacji. Na poziomie aplikacyjnym standaryzują sposób tworzenia aplikacji opartych na CORBA. Obiekty te dzielą się na niezależne od charakteru i zastosowania aplikacji – Horizontal CORBA Facilities, oraz na związane z dziedziną, w jakiej używana jest aplikacja – Vertical CORBA Facilities. Pierwsze z nich są bardziej

uniwersalne, drugie zaś są mocno związane z dziedziną, w jakiej jest stosowana konkretna aplikacja.

CORBA jest standardem określającym sposób funkcjonowania ORB oparty na współpracy klientów (usługobiorców) korzystających z usług obiektu (serwera – usługodawcy). Obiekt to element przetwarzający dane. Sposób jego implementacji oraz lokalizacja są ukryte przed klientem. Obiekt udostępnia swoje usługi poprzez interfejs. Klient tylko za pośrednictwem interfejsu może korzystać z tych usług. Wysyła on żądanie wykonania usługi korzystając z interfejsu obiektu. Żądanie przekazywane jest do ORB, a następnie przesyłane do obiektu. Usługa jest wykonywana i za pośrednictwem ORB zwracany jest wynik. Interfejs jest tworzony za pomocą języka IDL (Interface Definition Language). IDL jest językiem deklaratywnym, a nie jednym z języków programowania. Można w nim jedynie zapisać w sposób uniwersalny deklaracje usług obiektu. Tego typu podejście pozwala na implementację obiektów i klientów w różnych językach programowania. Aby zapewnić współpracę pomiędzy nimi, interfejsy tworzy się w IDL, a dopiero potem przekłada na język programowania, w jakich mają być zaimplementowane. W wyniku translacji powstaje po stronie klienta tzw. stub, a po stronie obiektu tzw. skeleton. Są to elementy łączące odpowiednio klienta i obiekty komunikujące je z ORB. W CORBA każdy obiekt ma swoją referencję. Poprzez nią klient odwołuje się do obiektu. Zarządzaniem referencjami zajmuje się ORB. Do niego należy przydzielenie jej obiektowi, przekazanie klientowi oraz utrzymywanie jej aktualności, nawet jeżeli obiekt z nią związany zmienia lokalizację. Elementem koordynującym współpracę ORB i obiektów jest tzw. Object Adapter. Jego zadaniem jest rejestrowanie obiektów, generowanie i interpretacja referencji do obiektów, aktywacja i dezaktywacja obiektów, wywoływanie metod obiektów. OMG wyspecyfikowała między innymi jeden taki obiekt, zwany BOA (Basic Object Adapter).

2.3. Zintegrowane środowisko JAVA / CORBA

Pierwsza wersja Javy nie przewidywała żadnego mechanizmu zdalnego umożliwiającego wywoływanie metod przez komponenty aplikacji rozproszonych. Obecną wersję 1.3 wyposażono w interfejs Java IDL (Interface Definition Language), który korzysta z architektury CORBA i pozwala na współpracę z dowolnym oprogramowaniem, zgodnym z tym standardem. Applety Javy mogą z powodzeniem wykorzystywać mechanizmy CORBY ze względu na ich komplementarność. Obie te technologie umożliwiają projektowanie i rozwijanie aplikacji internetowych w sposób nieporównywalnie lepszy od pozostałych technik. Wcześniej do Javy została wprowadzona metodologia RMI (Remote Method Invocation), która wspiera współdziałanie obiektów Javy wykonujących się na różnych maszynach wirtualnych.



Rys. 4. Scenariusz wykonania aplikacji internetowej w zintegrowanym środowisku Javy i CORBY

Fig. 4. Execution scenario of application WWW in integrated environment Java and CORBA

Rysunek 4 przedstawia sposób wykonania aplikacji internetowej, która po stronie klienta została zaimplementowana jako applet Javy, wykorzystujący mechanizmy CORBY dla wykonania operacji przez pozostałą na serwerze część komponentów aplikacji. Zastosowanie takiej techniki jest bardzo wygodne dla użytkownika. Eliminuje ona konieczność dystrybucji i ręcznej instalacji oprogramowania. Applet jest ściągany na komputer klienta przez przeglądarkę i uruchamiany przeważnie jako graficzny interfejs użytkownika danej aplikacji. Użytkownik za pomocą tego interfejsu wprowadza dane i wywołuje pewne zdarzenia. Część z nich może być obsłużona przez kod znajdujący się już po stronie klienta, jednak pozostałe obliczenia muszą być wykonane przez komponenty znajdujące się na serwerze. Odbywa się to poprzez wywoływanie metod za pomocą mechanizmu ORB. Zdalny obiekt wykonuje się na serwerze i przesyła do appletu wyniki swojego działania, które są wyświetlane za pomocą interfejsu graficznego. Oczywiście ten przykładowy scenariusz nie opisuje w pełni wszystkich możliwości tych technologii. Na przykład, wywołany zdalny obiekt może wywołać jako klient następny zdalny obiekt znajdujący się na jeszcze innym serwerze. Może się również zdarzyć, że obiekt działający na serwerze wywoła metodę appletu.

Podczas projektowania aplikacji internetowych pojawia się problem określenia architektury i rozlokowania komponentów na konkretnych komputerach (serwerach – klientach). Rozwiązanie tego problemu wiąże się z rozważeniem, jakie zadania muszą zostać wykonane lokalnie, a jakie zdalnie. Następnym krokiem jest zdefiniowanie interfejsów

```
package vics.corba.verscntl;

public class VerionControl
{
  ...
  public String getFileHistory(String file,
                               String revision,
                               String userName,
                               String userPassword)
    throws vics.corba.verscntl.FilenameException,
           vics.corba.verscntl.ReviosionException,
           vics.corba.verscntl.UserVerifyException,
           IE.Iona.Orbix2.CORBA.SystemException
  {
    ... // implementacja
  }
  ...
}
```

Rys. 5. Opis interfejsu i metody IDL w języku Java
Fig. 5. Description of the IDL interface and method in Java

pomiędzy wszystkimi komponentami. Wywoływanie metod na zdalnych obiektach odbywa się podobnie jak na lokalnych. Na rys. 5 przedstawiono definicję metody zaimplementowanej w języku Java. Metoda ta jest wywoływana przez obiekt klienta na zdalnym obiekcie, znajdującym się po stronie serwera. Wywołanie tej metody zobrazowane jest z kolei na rys. 6. Przedstawione operacje są możliwe dzięki wykorzystaniu mechanizmu Java IDL.

```
...
// uzyskanie referencji do obiektu „versionControl"
...
// wydobycie zestawienia historii pliku za pomocą
// metody zdalnego obiektu „versionControl"
try
{
  String tempFile = versionControl.getFileHistory(fileSpec.name(),
                                                  fileSpec.revision(),
                                                  context.user().name(),
                                                  context.user().password());
}
catch(vics.corba.verscntl.FilenameException e)
{
  // przechwycenie wszystkich możliwych błędów
  // wyspecyfikowanych w definicji metody
}
catch(vics.corba.verscntl.RevisionException
{
...
}
catch(...)
// ściągnięcie pliku z historią za pomocą ftp
...
```

Rys. 6. Wywołanie metody na zdalnym obiekcie
Fig. 6. Invocation of method on remote object

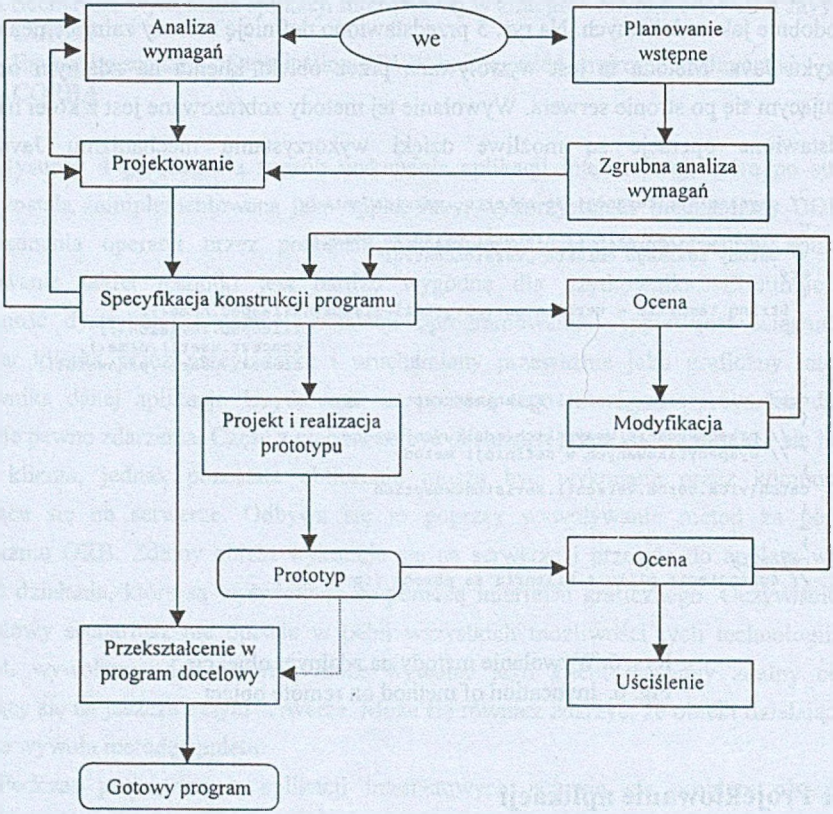
3. Projektowanie aplikacji

W związku z istnieniem zintegrowanych środowisk projektowania IDE (rozdział 1) oraz z dostępnością platform projektowania (rozdział 2) nasuwa się hierarchiczne podejście do

procesu wytwarzania aplikacji rozproszonych. Ogólną architekturę aplikacji (zestaw komponentów i środowisko ich wykonania) oraz szczegółowy opis wykorzystywanych komponentów określa się na podstawie analizy zdefiniowanych wymagań [6], przy czym analiza wymagań na ogół nie zależy od typu wytwarzanych aplikacji.

Przykładowy proces konstrukcji aplikacji mógłby odbywać się z uwzględnieniem następujących kroków:

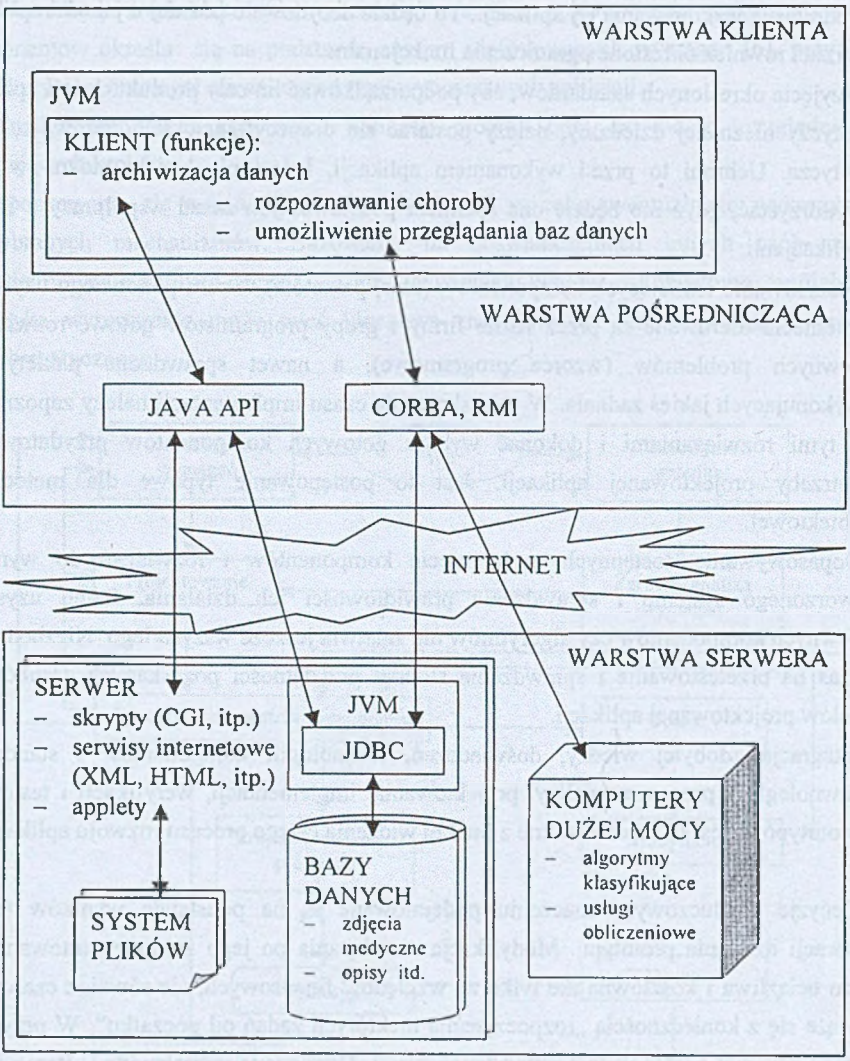
- Zapoznawanie się z istniejącymi rozwiązaniami w celu ewentualnego wykorzystania wybranych mechanizmów. Bazowanie na doświadczeniach innych osób ułatwia podejmowanie trafnych decyzji i eliminowanie wielu błędów. Jak wiadomo, zmniejszenie ryzyka wytwarzania może mieć kluczowe znaczenie dla realizacji danego projektu informatycznego.



Rys. 7. Model ewolucyjny procesu wytwarzania obiektowych aplikacji WWW
Fig. 7. Evolution developing process model for object web applications

- Określenie adekwatnej metodologii projektowania i implementacji oraz wybór środowiska wykonywania tej aplikacji. To będzie decydowało później o jakości aplikacji, narzuci również określone ograniczenia funkcjonalne.
- Przyjęcie określonych standardów, aby podporządkować im cały produkt. Jeżeli aplikacja dotyczy nieznanej dziedziny, należy postarać się o specyfikacje standardów, które jej dotyczą. Uchroni to przed wykonaniem aplikacji, której nie będzie można w pełni wykorzystać, gdyż nie będzie ona spełniała podstawowych zasad współpracy z innymi aplikacjami.
- Poszukiwanie istniejących komponentów (klas, pakietów) w celu ponownego użycia. W Internecie oferowane są przez różne firmy i grupy programistów gotowe rozwiązania pewnych problemów (wzorce programowe), a nawet sprawdzone pakiety klas wykonujących jakieś zadania. W celu skrócenia czasu implementacji należy zapoznać się z tymi rozwiązaniami i dokonać wyboru gotowych komponentów przydatnych na potrzeby projektowanej aplikacji. Jest to postępowanie typowe dla metodologii obiektowej.
- Dopasowywanie dostępnych w Internecie komponentów i rozwiązań do wymagań tworzonego systemu i sprawdzenie prawidłowości ich działania. Samo uzyskanie pewnych komponentów czy algorytmów nie załatwia jeszcze wszystkiego. Niezbędny jest czas na przetestowanie i sprawdzenie stopnia przydatności pozyskanych zasobów dla celów projektowanej aplikacji.
- Integracja zdobytej wiedzy, doświadczeń, technologii, komponentów i standardów równolegle z procesem analizy, projektowania, implementacji, weryfikacji i testowania prototypów. Jest to bardzo istotne z punktu widzenia całego procesu rozwoju aplikacji.

Decyzje o kluczowym znaczeniu podejmowane są na podstawie wyników testów, obserwacji działania prototypu. Modyfikacja rozwiązania po jego zaimplementowaniu jest bardzo uciążliwa i kosztowna nie tylko ze względów finansowych, ale również czasowych, bo wiąże się z koniecznością „rozpoczynania niektórych zadań od początku". W przypadku ogólnym schemat postępowania przedstawia rys. 7. Odpowiada interakcyjno – iteracyjnemu modelowi wytwarzania [1] i prowadzi do trójwarstwowej struktury aplikacji przedstawionej na rys. 7. Szczegóły takiego rozwiązania podane są w rozdziale następnym. W strukturze tej uwzględniono już wpływ platformy Java i środowiska CORBA. Najważniejszymi problemami są: poszukiwanie odpowiednich komponentów, ich przystosowanie (dopasowanie) i integracja, a także wybór odpowiednich interfejsów i obiektów pozwalających na stworzenie oprogramowania wysokiej jakości. Szczegółowe rozwinięcie tej struktury jest możliwe dla konkretnie realizowanej aplikacji.

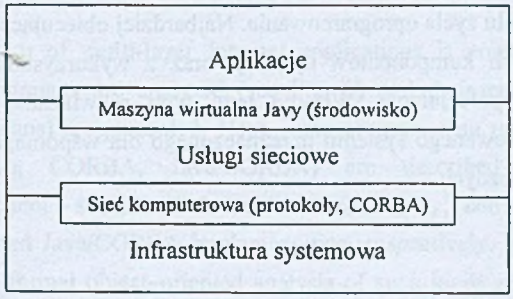


Rys. 8. Trójwarstwowy model klient – serwer Endoskopowego Systemu Rekomendacji

Fig. 8. 3-Tier Client/Server model of Endoscopy Recommender System

4. Przykład trójwarstwowej aplikacji rozproszonej

Jako przykład wytwarzania aplikacji rozproszonej przyjęto system ERS (Endoscopy Recommender System) projektowany w ramach współpracy Katedry Architektury Systemów Komputerowych Politechniki Gdańskiej z Kliniką Gastroenterologiczną Akademii Medycznej w Gdańsku. System ten ma na celu archiwizowanie badań endoskopowych pacjentów oraz wspomaganie lekarza w rozpoznawaniu schorzeń gastroenterologicznych. Dla każdego pacjenta tworzy się odpowiedni rekord badań w standardzie DICOM. Rekordy te, zawierające wywiad z pacjentem, wyniki badań, rozpoznanie i leczenie, przechowywane są w bazie danych, która jest dołączona do Internetu. Dzięki odpowiedniej przeglądarce możliwe jest wyszukiwanie zdjęć z konkretnych badań oraz ich szczegółowa analiza. W tym celu wykorzystuje się zaawansowane algorytmy klasyfikujące. Istnieje również możliwość porównania zdjęć lub odpowiednich obiektów na tych zdjęciach i na tej podstawie zasygnalizowanie ujęcia lub ich sekwencji, które mogą zaświadczyć o danej jednostce chorobowej. Trójwarstwową architekturę projektowanego systemu przedstawia rys. 9.



Rys. 9. Umiejscowienie aplikacji internetowych w systemach komputerowych
Fig. 9. Place of Web applications in computer systems

Uwzględnia ona możliwości i ograniczenia platformy programowania Javy i środowiska CORBA bez wykorzystania zintegrowanego środowiska IDE. W skład aplikacji wchodzą następujące komponenty:

- przeglądarka DICOM,
- baza danych rekordów pacjenta w formacie DICOM,
- serwer DICOM,
- moduł analizatora obrazów,
- moduł wspomagający rozpoznawania chorób.

Implementacja tych komponentów jest sporym wyzwaniem, przy czym szczególną uwagę zwraca się na właściwą konstrukcję bazy danych i analizatora obrazów, a także sposób koordynacji pracy wszystkich komponentów.

Na serwerach sieciowych posadowione są na ogół takie serwisy, jak np. serwis stron WWW, ftp, czy tzw. mirrory itp. Znajdują się tam też skrypty napisane przy użyciu CGI, czy Perla, a także servlety opisane w Javie. Zainstalowane są również bazy danych dostępne poprzez protokoły JDBC (Java DataBase Connection). Dostęp do tego typu usług jest możliwy przy wykorzystaniu protokołów: Java API, CORBA i RMI, które należą do warstwy pośredniej.

5. Uwagi końcowe

W pracy przedstawiono problematykę wytwarzania aplikacji rozproszonych w środowisku Java / CORBA w oparciu o model wielowarstwowy. Istniejące zintegrowane środowiska tego typu nie są jeszcze w pełni dojrzałe. Trudno też wykorzystać technologie obiektowe w całym cyklu życia oprogramowania. Najbardziej obiecujące podejście wiąże się z określeniem głównych komponentów aplikacji oraz z wykorzystaniem standardowych metod integracji i współdziałania. Właśnie tego typu rozwiązanie zastosowano przy projektowaniu specjalizowanego systemu przeznaczonego dla wspomagania rozpoznawania chorób gastroenterologicznych.

LITERATURA

1. Bobkowska A., Krawczyk H.: Wytwarzanie sieciowych aplikacji użytkowych z wykorzystaniem notacji UML. Zeszyty Naukowe Politechniki Śląskiej, Seria: Informatyka z. 36, Gliwice 1999, s. 49 – 64.
2. Evans E., Rogers D.: Using Java Applets and CORBA for Multi-User Distributed Applications. IEEE Internet Computing, May –June 1997, pp. 43 – 55.
3. Gosling J., Jay B., Steele G.: The Java Language Specification. Addison Wesley, Reading, March, 1996.
4. Kruger D., Adler R.M.: The Emergence of Distributed Component Platforms. IEEE Computer, March 1998, pp. 44 – 53.
5. Krzyżek A.: Usługi w sieci Internet. Wyd. Krakowskie Centrum Informatyki Stosowanej, Kraków 1998.

6. Loucopoulus P., Karakostas V.: Systems Requirements Engineering. McGraw-Hill Inc., 1995.
7. Manola F.: Technologies for Web Object Model. IEEE Internet Computing, Jan – Feb, 1999, pp. 38 – 47.
8. Rumbaugh J., Blaha M., Premerlani W., Eddy S. and Lorensen W.: Object-oriented Modeling and Design. Prentice-Hall, 1991.
9. Siegel J.: A Preview of CORBA 3. IEEE Computer, May 1999, pp. 114 – 116.
10. The Jini (TM) Technology - http://www.sun.com/jini/

Recenzent: Dr inż. Przemysław Szmal

Wpłynęło do Redakcji 27 marca 2000 r.

Abstract

A design problem of multi-layer Internet applications is considered. The integrated development environments are analysed (see Fig. 1) and their complete implementation (DCOM and JavaBeans) are presented. Next, main programming platforms and computing environments (Java, CORBA, Java/CORBA) are described and definition and implementation of client – server models are discussed. Fig. 2, 3 and 4 present such solutions for Java, CORBA and Java/CORBA implementation, respectively. It was shown that there was a gap between formal object–oriented analysis of such kinds of applications and their implementation of such environments. Basing on these, main problems of development phases are listed and schema of interactive – iterational approach is proposed (see Fig. 7). How such methodology impacts on multi-layer architectures is shown in Fig. 8. Then on this base the 3-tier architecture for Internet Endoscopy Recommender System is suggested. Such a medical system can support diagnosis on the base of endoscopy image recognition. The suitable patient records are stored in a multimedia database and can be accessed by DICOM-oriented applications.