

XIII Konferencja PLOUG
Kościelisko
Październik 2007

Charakterystyka techniczna ADF Faces 11g Rich Client

Maciej Zakrzewicz
PLOUG, Politechnika Poznańska

Abstrakt. ADF Faces 11g Rich Client to nowa wersja biblioteki komponentów wizualnych do wykorzystania w aplikacjach powstających w oparciu o szkielet JavaServer Faces (Java EE). Na szczególną uwagę zasługuje bogactwo funkcjonalne oraz wykorzystanie technologii AJAX w celu poprawy interakcyjności komponentów wizualnych. Celem referatu jest ocena funkcjonalności i efektywności ADF Faces 11g Rich Client.

Informacje o autorze. Pracownik Instytutu Informatyki Politechniki Poznańskiej oraz prezes Zarządu Stowarzyszenia Polskiej Grupy Użytkowników Systemu Oracle. Zainteresowania naukowe obejmują eksplorację danych (data mining), systemy baz danych/hurtowni danych oraz technologie internetowe. Dla krajowych i zagranicznych uniwersytetów oraz przedsiębiorstw (m.in. Niemcy, Wielka Brytania, USA) prowadzi wykłady i szkolenia z zakresu projektowania i implementacji systemów informatycznych. Kieruje i doradza w projektach informatycznych realizowanych w architekturach internetowych. Autor wielu publikacji z zakresu odkrywania zbiorów częstych, przetwarzania zapytań eksploracyjnych, adaptatywnych serwerów WWW, technik indeksowania baz danych, strojenia systemów baz danych, standardów Java dla baz danych i dla aplikacji wielowarstwowych. Współtwórca serii konferencji i seminariów PLOUG, służących transferowi wiedzy i technologii w środowisku krajowych przedsiębiorstw informatycznych.

1. Wprowadzenie

Już od ponad 10 lat architektura Java Enterprise Edition (Java EE) stanowi pokusę dla projektantów i programistów systemów internetowych zainteresowanych realizacją komponentowych, przenaszalnych, efektywnych aplikacji dla baz danych. Rozwiązania J2EE bazują na dynamicznym generowaniu graficznego interfejsu użytkownika przeglądarki (np. HTML) za pomocą kodu wyrażonego w języku Java - języku szczytującym się bezprecedensowo ogromną liczbą bibliotek standardowych, m.in. służących do współpracy z bazami danych. Pomimo względnej młodości technologii Java EE, w minionych latach mogliśmy już zaobserwować jej liczne ewolucje i metamorfozy. Pierwsze systemy aplikacji na platformie Java EE były oparte głównie o serwlety Java, stanowiące pochodną prostej koncepcji CGI, implementowane w formie pojedynczych klas Java odpowiedzialnych za generowanie kompletnych dokumentów WWW. Brak naturalnych mechanizmów oddzielenia kodu logiki biznesowej od kodu prezentacji danych był głównym źródłem krytyki tego rozwiązania technicznego. W niedługim czasie alternatywą dla tradycyjnej metody implementacji serwletów stał się mechanizm stron JavaServer Pages (JSP), umożliwiających konstrukcję szablonów dokumentów WWW, zawierających zanurzony kod Java odpowiedzialny wyłącznie za generowanie dynamicznych fragmentów dokumentu. Równocześnie wiele uwagi poświęcano zagadnieniom implementacji logiki biznesowej, proponując przede wszystkim technologię Enterprise JavaBeans, w ramach której programista implementował funkcje biznesowe lub funkcje dostępu do danych w formie klas Java z interfejsami zdalnego dostępu.

Celem tego artykułu jest charakterystyka nowych rozwiązań technicznych, które zostały wprowadzone w wersji 11g biblioteki ADF Faces, będącej propozycją Oracle służącą uproszczeniu konstrukcji graficznego interfejsu użytkownika w aplikacjach Java EE. Struktura tekstu jest następująca. Rozdział 2 omawia podstawowe architektury aplikacji Java EE. Rozdział 3 jest wprowadzeniem do technologii ADF Faces. W rozdziale 4 omówiono główne założenia technologii AJAX. Rozdział 5 przedstawia nowe własności biblioteki ADF Faces Rich Client. Rozdział 6 jest podsumowaniem.

2. Architektury aplikacji Java EE

Wybór odpowiedniej architektury dla nowoprojektowanej aplikacji Java EE ma kluczowe znaczenie z punktu widzenia efektywnego zarządzania projektem, łatwości późniejszej pielęgnacji aplikacji, elastyczności wdrażania, przyszłej integrowalności z innymi systemami. W przeszłości najczęściej proponowano dwuwarstwowe modele architektury, w których kod aplikacyjny był dzielony na logikę prezentacji i logikę biznesową. Logika prezentacji odpowiadała za koordynację obsługi żądań, walidację danych, wizualizację danych i interakcję z użytkownikiem. Logika biznesowa odpowiadała za przetwarzanie danych i interakcję z warstwą danych. Implementacja logiki prezentacji odbywała się z użyciem technologii serwletów Java i JavaServer Pages, natomiast logiki biznesowej – z użyciem technologii JavaBeans lub Enterprise JavaBeans.

Obecnie dominującym wzorcem architektury dla aplikacji Java EE jest Model-View-Controller (MVC), który zakłada trójwarstwowy podział kodu aplikacyjnego. Warstwa modelu (Model) odpowiada za przetwarzanie danych i interakcję z bazą danych, merytorycznie zatem jest równoważna warstwie logiki biznesowej w modelu dwuwarstwowym. Warstwa prezentacji (View) odpowiada za wizualizację danych, ale już nie za interakcję z użytkownikiem, ani za walidację danych, ani za koordynację obsługi żądań. Różni się zatem istotnie od swojego dwuwarstwowego odpowiednika w postaci logiki prezentacji. Nowa warstwa, warstwa kontrolera (Controller) odpowiada właśnie za interakcję z użytkownikiem, walidację danych wejściowych i koordynuje obsługę żądań dokonując stosownych wywołań komponentów prezentacji i modelu.

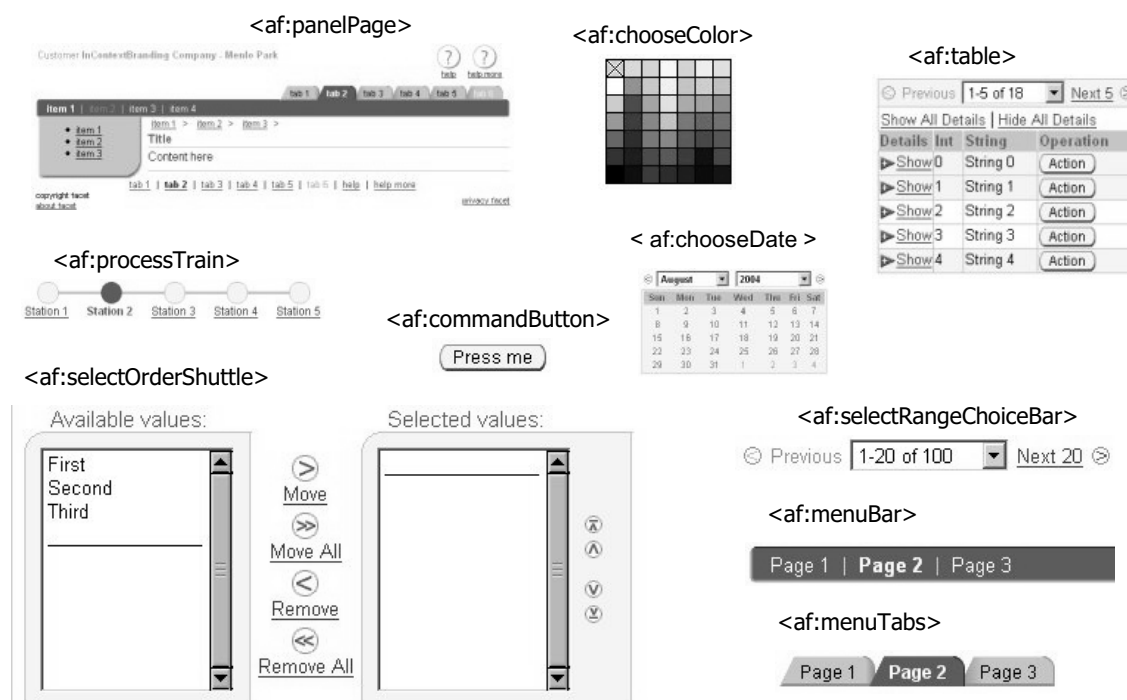
Stosowanie wzorca projektowego Model-View-Controller umożliwia sprawny podział ról w zespole projektowym, ułatwia reorganizację procesów biznesowych implementowanych w aplikacji, a ponadto zwiększa szanse na powtórne wykorzystanie tworzonych modułów w innych systemach aplikacyjnych. Nie należy jednak ukrywać faktu, iż tworzenie aplikacji Java EE zgodnie z tą architekturą jest trudniejsze od konstrukcji aplikacji monolitycznych lub dwuwarstwowych. Stąd wynika silne zapotrzebowanie na zaawansowane narzędzia i środowiska programistyczne, które ułatwiają zarządzanie realizowanymi projektami.

Dotąd w świecie Java EE swoją obecność zaznaczyły dwie technologie wspomagające tworzenie aplikacji w architekturze Model-View-Controller: Apache Struts i JavaServer Faces. Apache Struts, jako pierwsze na rynku, zaoferowały programiście Java EE gotowy moduł konfigurowalnego kontrolera, koncepcję reprezentacji stanu aplikacji za pomocą obiektów Java automatycznie tworzonych przez kontroler i przekazywanych kolejnym warstwom (Form Beans), zaawansowane biblioteki znaczników służących interakcji pomiędzy warstwami i internacjonalizacji aplikacji. JavaServer Faces to z kolei rozwiązanie bazujące na funkcjonalności Apache Struts, ale jednocześnie oferujące interesujące mechanizmy konstrukcji graficznego interfejsu użytkownika za pomocą bibliotek znaczników. Niestety, zarówno Apache Struts, jak i JavaServer Faces nie oferowały programistom satysfakcjonujących rozwiązań w zakresie implementacji warstwy modelu.

3. Wprowadzenie do ADF Faces

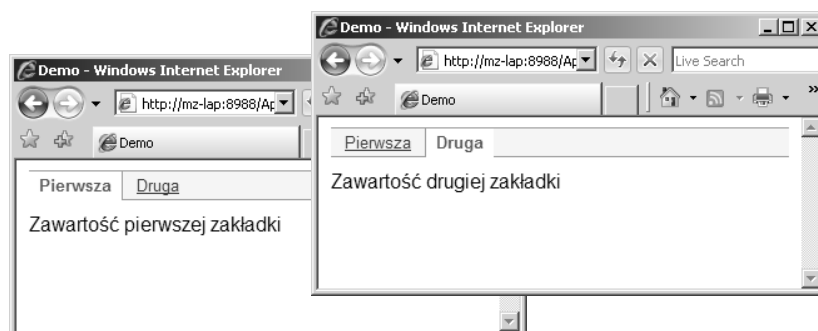
Firma Oracle dostrzega w technologii Java EE następcę dla promowanych przez siebie w przeszłości platform Oracle Forms. Stąd obserwuje się wiele intensywnych prac nad dostarczeniem programistom skutecznych narzędzi i bibliotek, za pomocą których będą mogli uzyskać podobną produktywność i wydajność pracy jak w przeszłości, kiedy to budowali swoje aplikacje wyłącznie w technologii Oracle Forms. Główne efekty tych prac to: dynamiczny rozwój narzędzia projektowo-programistycznego Oracle JDeveloper, rozwój technik współpracy z bazami danych (EJB, TopLink, ADF Business Components), biblioteki komponentów wizualnych (ADF Faces), rozbudowa funkcjonalności warstwy kontrolera JSF (ADF Bindings).

Jednym z ciekawszych rozwiązań proponowanych przez Oracle w ramach rodziny ADF (Application Development Framework) jest biblioteka komponentów wizualnych ADF Faces 10g, ułatwiająca programistom implementację graficznego interfejsu użytkownika w oparciu o JavaServer Pages/JavaServer Faces. Korzyści wynikające z stosowania ADF Faces 10g są dwutorowe. Po pierwsze, wiele tradycyjnych elementów wizualnych HTML zostało zastąpionych odpowiednikami o większej funkcjonalności, np. pole tekstowe może posiadać etykietę, samodzielnie walidować wprowadzane dane, a jego zdarzenia mogą być obsługiwane za pomocą języka Java, a nie wyłącznie JavaScript. Po drugie, ADF Faces 10g dostarcza szereg zupełnie nowych elementów wizualnych, które nie mają swoich odpowiedników w HTML, np. kalendarz do wyboru daty wprowadzanej do pola tekstowego. Przykłady nowych komponentów wizualnych ADF Faces 10g przedstawiono na rys. 1.



Rys. 1. Wybrane komponenty ADF Faces 10g nie posiadające odpowiedników w HTML

Wykorzystywanie komponentów wizualnych ADF Faces polega na dołączeniu dwóch bibliotek znaczników do tworzonej strony JSP, a następnie osadzeniu na stronie komponentów za pomocą znaczników pochodzących z tych bibliotek. Zachowanie się każdego z komponentów jest definiowane przez zbiór właściwości zapisywanych jako atrybuty znaczników. Niektóre z komponentów wymagają implementacji po stronie serwera kodu w języku Java, np. kodu walidującego. Przykład prostego użycia komponentów ADF Faces przedstawiono na rys. 2, gdzie znaczniki „showOneTab” i „showDetailItem” zastosowano do konstrukcji dwuzakładowej strony JSP.



```
<%@ taglib uri="http://xmlns.oracle.com/adf/faces" prefix="af"%>
...
<af:showOneTab position="above">
  <af:showDetailItem text="Pierwsza">
    <af:outputText value="Zawartość pierwszej zakładki"/>
  </af:showDetailItem>
  <af:showDetailItem text="Druga">
    <af:outputText value="Zawartość drugiej zakładki"/>
  </af:showDetailItem>
</af:showOneTab>
```

Rys. 2. Przykład użycia komponentów ADF Faces

4. Wprowadzenie do AJAX

Wrodzoną wadą technologii Java EE, poważnie utrudniającą realizację wysoce interaktywnych aplikacji biznesowych, było oparcie obsługi żądań użytkownika na znanym ze specyfikacji protokołu HTTP modelu żądanie-odpowiedź. Model ten zakłada, że użytkownik wyraża swoje żądanie wprowadzając adres URL lub klikając interakcyjny element graficznego interfejsu, a aplikacja Java EE odpowiada na to żądanie generując kompletną stronę HTML do wyświetlenia na ekranie przeglądarki. Konsekwencją przyjęcia takiego założenia są m.in. problemy z: (1) bieżącą walidacją danych, gdyż walidacja może być wykonana przez aplikację Java EE dopiero po zatwierdzeniu kompletnego formularza przez użytkownika, (2) konstrukcją formularzy typu nadrzędny-podrzędny (Master-Detail), gdyż formularze takie muszą z góry otrzymać komplet danych do prezentacji, (3) obsługą zdarzeń graficznego interfejsu użytkownika, gdyż musi ona być implementowana w języku JavaScript.

Rozwiązaniem tych fundamentalnych problemów może być zyskująca obecnie na popularności technologia AJAX (Asynchronous JavaScript and XML), która umożliwia dokumentowi HTML wyświetlonemu w przeglądarce nawiązywanie w tle połączeń ze zdalnym serwerem aplikacji, pobieranie nowych danych oraz dynamiczną przebudowę swojej zawartości. Dzięki zastosowaniu AJAX, przeglądarka może dostarczać użytkownikowi nowe treści bez potrzeby pobierania/przeładowywania i ponownego wyświetlania kompletnej strony HTML. Komunikacja z serwerem aplikacji odbywa się asynchronicznie, za pomocą ramek XML przekazywanych przez protokół HTTP. Mechanizmy AJAX są programowane w języku JavaScript. W celu umożliwienia programistom korzystania z AJAX, przeglądarki udostępniają komunikacyjny obiekt JavaScript o nazwie XMLHttpRequest. Przykład użycia tego obiektu z poziomu języka JavaScript przedstawiono na rys. 3. Znaczenie przedstawionego kodu jest następujące. W odpowiedzi na działanie użytkownika, pobierany jest uchwyt do obiektu XMLHttpRequest (implementacja zależna od rodzaju przeglądarki), a następnie za pomocą metod „open” i „send” wysyłane jest w tle żądanie do serwera aplikacji. W przedstawionym przykładzie jest to żądanie pobrania obiektu o adresie URL „hello.txt”. Ponadto, tuż przed wysłaniem żądania, wskazywana jest metoda odpowiedzialna za późniejszy odbiór odpowiedzi na to żądanie. Na rysunku przedstawiono także kod metody „myHandler”, która zostanie automatycznie wywołana przez przeglądarkę gdy tylko nadejdzie odpowiedź na żądanie. W przedstawionym przykładzie metoda ta dołącza otrzymany obiekt do istniejącego na stronie elementu HTML o identyfikatorze „ID1”. W ten sposób bez potrzeby przeładowywania strony HTML, jeden z jej elementów wyświetli treść pobraną w tle od serwera aplikacji. Podkreślmy wagę asynchronizmu komunikacji AJAX: pomiędzy wysłaniem żądania a otrzymaniem odpowiedzi przeglądarka funkcjonuje bez zakłóceń i realizuje ewentualne pozostałe życzenia użytkowników.

```
if (window.XMLHttpRequest) {
    requester = new XMLHttpRequest();
}
else
    if (window.ActiveXObject) {
        requester = new ActiveXObject("Microsoft.XMLHTTP");
    }
requester.onreadystatechange = myHandler;
requester.open("GET", "hello.txt");
requester.send(null);

function myHandler()
{
    if (requester.readyState == 4) {
        if (requester.status == 200) {
            el = document.getElementById("ID1");
            el.innerHTML += requester.responseText;
        }
    }
    return true;
}
```

Rys. 3. Przykład użycia technologii AJAX

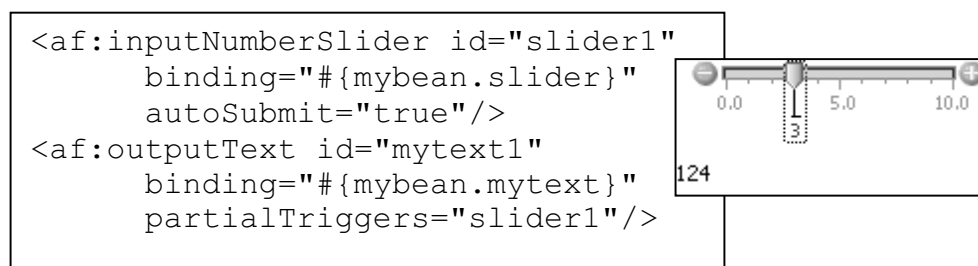
5. ADF Faces Rich Client (11g)

W ramach stopniowego rozwoju funkcjonalności biblioteki komponentów wizualnych ADF Faces, już w wersji 10g tej biblioteki Oracle proponował stosowanie mechanizmów w stylu AJAX nie opierając się jednak na oryginalnej specyfikacji AJAX (tzw. AJAX without AJAX). W zapowiadanej wersji 11g elementy ADF Faces w pełni wykorzystują AJAX do interakcji ze stroną serwera aplikacji. W celu wyróżnienia tego rozwiązania, Oracle nazywa nową wersję biblioteki „ADF Faces Rich Client”.

Komponenty ADF Faces Rich Client umożliwiają programiście implementację natychmiastowej obsługi zdarzeń generowanych przez komponent za pomocą kodu Java znajdującego się po stronie serwera aplikacji, zamiast za pomocą kodu JavaScript znajdującego się po stronie przeglądarki. Gdy komponent ADF Faces Rich Client generuje zdarzenie, nawiązuje połączenie AJAX z serwerem aplikacji i przekazuje mu opis zdarzenia, wówczas serwer aplikacji wykonuje kod obsługi zdarzenia i przekazuje przeglądarce informacje o zmianie stanu aplikacji. Przeglądarka, po otrzymaniu informacji o zmianie stanu aplikacji, aktualizuje wygląd graficznego interfejsu użytkownika. Cały proces obsługi zdarzenia odbywa się bez potrzeby przeładowywania aktualnie wyświetlanej strony. Mechanizmy te zostały nazwane Partial Submit i Partial Page Rendering – pierwszy oznacza przekazanie serwerowi aplikacji opisu zdarzenia bez potrzeby zatwierdzania całego formularza, drugi oznacza aktualizację treści strony bez potrzeby jej przeładowania. Aktywacja i konfiguracja tych mechanizmów odbywa się za pomocą dwóch atrybutów znaczników komponentów: „autoSubmit” i „triggers”.

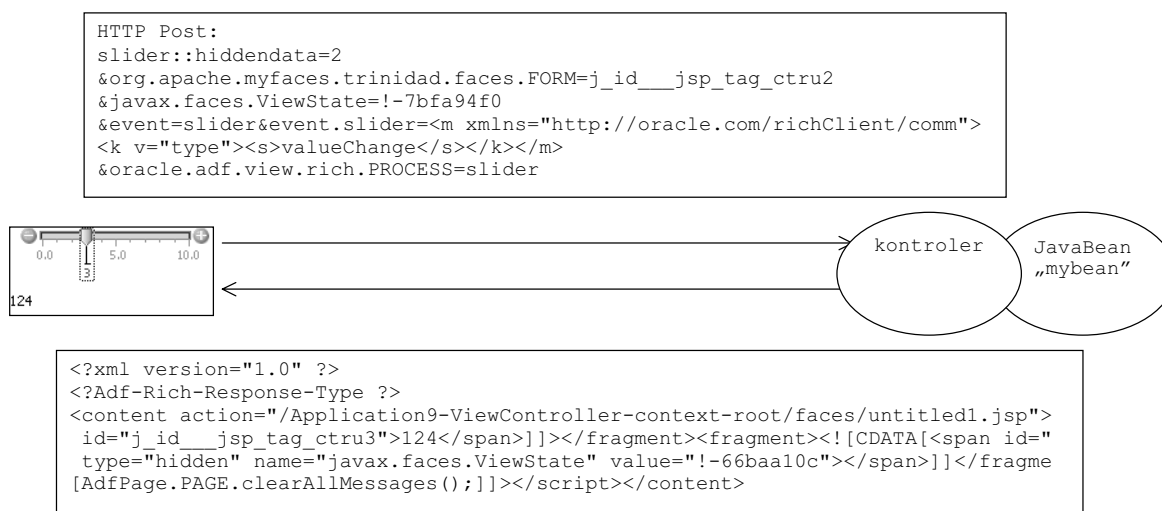
Przeanalizujemy następujący przykład działania mechanizmów Partial Submit i Partial Page Rendering. Na rys. 4 znajduje się kod źródłowy strony JSP zawierającej dwa elementy wizualne ADF Faces: suwak liczbowy i pole tekstowe. Naszym celem będzie obsługa zmiany położenia suwaka poprzez wykonanie kodu Java znajdującego się po stronie serwera aplikacji, a następnie wyświetlenie wyniku obliczeń w polu tekstowym. W tym celu po stronie serwera aplikacji stworzymy obiekt JavaBean posiadający atrybuty „slider” i „mytext” a także cztery metody dostępne: „getSlider()”, „setSlider()”, „getMytext()” i „setMyText()”. Obiekt ten nazywa się „mybean” i jest zdefiniowany w plikach konfiguracyjnych po stronie serwera aplikacji. Załóżmy, że po zmianie wartości atrybutu „slider” za pomocą metody „setSlider()” następuje przeliczenie/aktualizacja

wartości atrybutu „mytext”. Aby umożliwić suwakowi interakcję z serwerem aplikacji po każdej zmianie położenia, do znacznika wprowadzono atrybut „autoSubmit” o wartości „true”, a za pomocą atrybutu „binding” wskazano obiekt i atrybut obiektu JavaBean odpowiedzialnego za obsługę zmiany stanu suwaka. Natomiast w celu umożliwienia automatycznej aktualizacji wartości pola tekstowego po zmianie położenia suwaka, do jego znacznika wprowadzono atrybut „partialTriggers” identyfikujący elementy, których zdarzenia mogą mieć wpływ na wartość wyświetlaną przez to pole tekstowe. Ponadto, powiązano treść pola tekstowego z atrybutem obiektu JavaBean (atrybut „binding”).



Rys. 4. Przykład kodu źródłowego strony wykorzystującej Partial Submit i Partial Page Rendering

Na rys. 5 przedstawiono przebieg interakcji pomiędzy przeglądarką a serwerem aplikacji w chwili, gdy element graficznego interfejsu użytkownika generuje zdarzenie. Element nawiązuje połączenie AJAX z serwerem aplikacji i przesyła informację o swoim zdarzeniu („valueChange”). W odpowiedzi, serwer aplikacji wykonuje skojarzony z elementem kod metody „setSlider()”. Skutkiem tego wywołania jest zmiana stanu aplikacji – wartości atrybutu skojarzonego z polem tekstowym. Serwer aplikacji wysyła wówczas do przeglądarki dokument XML zawierający opis zmiany stanu aplikacji (wartość „124” dla pola tekstowego). Przeglądarka dokonuje dynamicznej modyfikacji pola tekstowego wyświetlanego na ekranie bez potrzeby przeładowania całego dokumentu HTML. Ponadto, dotychczasowy stan suwaka nie jest tracony.

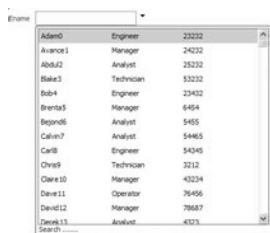


Rys. 5. Interakcja pomiędzy komponentem ADF Faces a serwerem aplikacji

Oprócz wbudowanej implementacji mechanizmów AJAX, biblioteka ADF Faces Rich Client oferuje programistom także wiele innych nowości. Powiększono zbiór podstawowych elementów wizualnych, m.in. o wielokolumnowe listy rozwijane, listy wartości związane z polami tekstowymi, pola tekstowe z przyciskami inkrementacji/dekrementacji, suwaki liczbowe i czasowe, panele wyszukiwania. Wygląd tych elementów został przedstawiony na rys. 6. Pojawiły się również elementy wizualne służące do reprezentacji wykresów. Umożliwiono programistom implementację

walidacji po stronie klienta, a także rozbudowano mechanizmy konfiguracji modułu kontrolera (Task Flow).

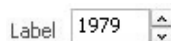
`<af:inputComboboxListOfValues>`



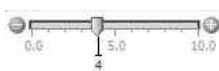
`<af:inputListOfValues>`



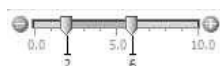
`<af:inputNumberSpinbox>`



`<af:inputNumberSlider>`



`<af:inputRangeSlider>`



`<af:query>`



Rys. 6. Wybrane nowe elementy wizualne biblioteki ADF Faces Rich Client

6. Podsumowanie

Nowa biblioteka komponentów wizualnych ADF Faces Rich Client jest krokiem we właściwym kierunku, w stronę zapewnienia programistom aplikacji Java EE podobnych warunków pracy, jak znane z klasycznych środowisk Borlanda czy Microsoftu. Obecność mechanizmów AJAX sprawia, że eliminuje się podstawową wadę architektury aplikacji internetowych, jaką jest funkcjonowanie modelu żądanie-odpowiedź. Programista zyskuje narzędzie, za pomocą którego tworzy aplikację o rozbudowanym graficznym interfejsie wyświetlanym po stronie przeglądarki, lecz wspieranym przez aktywny kod Java znajdujący się po stronie serwera aplikacji. Interakcja pomiędzy tym graficznym interfejsem a kodem Java odbywa się w sposób niewidoczny dla użytkownika (a nawet dla programisty), w tle.

Należy jednocześnie zdawać sobie sprawę z zagrożeń, jakie mogą wynikać ze stosowania tego nowego modelu interakcji. Po pierwsze, o ile w klasycznych aplikacjach Java EE obsługa jednej strony JSP wymagała przesłania jednego sieciowego żądania i jednej odpowiedzi, o tyle w aplikacjach opartych o AJAX takich przesłań może być dość dużo (oddzielne przesłanie sieciowe dla każdego wygenerowanego zdarzenia), co niewątpliwie przełoży się na większą konsumpcję pasma sieciowego. Po drugie, komunikacja AJAX ma charakter asynchroniczny, co przy opóźnieniach sieciowych i opóźnieniach wynikających z przetwarzania żądań przez serwer aplikacji, może prowadzić do „zagubienia się” użytkownika aplikacji, który z opóźnieniem będzie świadkiem zmian stanu aplikacji (np. walidacja pola zgłasza komunikat błędu po kilku sekundach od opuszczenia pola). Po trzecie, wzmożona interakcja sieciowa pomiędzy przeglądarką a serwerem aplikacji prowadzi do zwiększenia obciążenia serwera, co może przełożyć się na jego wyższe wymagania sprzętowe.

Bibliografia

- [1] „Oracle ADF Faces”, dokumentacja techniczna, Oracle
- [2] “Hello Ajax! How to Do the Ajax Thing with Oracle JDeveloper”, Frank Nimphius
- [3] “The Benefits Of The AJAX RenderKit”, Jonas Jacobi, John Fallows