Oracle ADF i JBoss Seam - dwa skrajnie różne podejścia do współpracy JSF z EJB

Bartosz Mordaka, Marek Wojciechowski
Politechnika Poznańska
e–mail: Marek.Wojciechowski@cs.put.poznan.pl

Abstrakt. JavaServer Faces (JSF) i Enterprise JavaBeans (EJB) to dwie kluczowe technologie platformy Java EE. Pierwsza z nich jest obecnie podstawową technologią do implementacji warstwy prezentacji w aplikacjach Java EE, a druga od lat ma status "oficjalnej" technologii dla warstwy logiki biznesowej. Niniejszy artykuł poświęcony jest problematyce współpracy JSF i EJB w aplikacjach Java EE. Jak dotąd ta współpraca ciągle nie doczekała się standaryzacji, dlatego programiści zdani są na własne rozwiązania lub te oferowane przez istniejące szkielety aplikacji. Celem artykułu jest przedstawienie i porównanie podejść do integracji JSF i EJB w ramach szkieletów aplikacji Oracle ADF i JBoss Seam.

1. Wprowadzenie

Java Platform, Enterprise Edition (w skrócie: Java EE) [JEE] to jedna z dwóch (obok Microsoft .NET) najważniejszych platform do tworzenia złożonych, komponentowych aplikacji biznesowych. Java EE jako standard ma postać zbioru specyfikacji technologii składowych, poświęconych różnym obszarom funkcjonalnym aplikacji. Java EE jest od początku swojego istnienia silnie zorientowana na wielowarstwową architekturę aplikacji, ze szczególnym uwzględnieniem rozbicia aplikacji pracującej na serwerze aplikacji na warstwy prezentacji i logiki biznesowej.

Technologie prezentacji platformy Java EE to serwlety i JavaServer Pages (JSP) wraz z technologią szkieletową (ang. framework) JavaServer Faces (JSF), systematyzującą sposób wykorzystania serwletów i JSP w aplikacji webowej zgodnie z wzorcem projektowym Model-View-Controller (MVC). Z myślą o implementacji logiki biznesowej, platforma Java EE oferuje technologię Enterprise JavaBeans (EJB). W swoich wczesnych wcieleniach technologia EJB była nienaturalna, skomplikowana, a do tego często prowadząca co mało wydajnych i słabo skalowalnych aplikacji. Należy tu wspomnieć, że właśnie na krytyce EJB [JoHo04] wyrósł najpopularniejszy framework dla Javy o nazwie Spring [Spring], opierający logikę biznesową o zwykłe obiekty Java (ang. POJO – Plain Old Java Object). Jednakże w wersji 3.0 technologia EJB została uproszczona i odchudzona w sposób odpowiadający aktualnym trendom. Zredukowano liczbę plików źródłowych składających się na definicję komponentu EJB, architekturę oparto o zwykłe klasy i interfejsy (POJO i POJI), deskryptory XML-owe zastąpiono adnotacjami, wykorzystano wstrzykiwanie zależności (ang. dependency injection) do dostępu do komponentów. Ponadto, wydzielono z technologii EJB funkcjonalność odpowiedzialną za reprezentację danych z bazy danych, ustanawiając nowy standard Java Persistence API (JPA), oparty o sprawdzone w praktyce mechanizmy odwzorowania obiektowo-relacyjnego. EJB 3.0 jest więc dzisiaj „domyślną opcją" w zakresie implementacji logiki biznesowej na platformie Java EE, szczególnie dla aplikacji wykorzystujących transakcje lub posiadających zarówno klientów webowych jak i aplikacyjnych.

Java EE w wersji 5 z jednej strony jednoznacznie wskazuje na użycie JSF w warstwie prezentacji, a EJB w warstwie usług biznesowych, z drugiej strony nie standaryzuje, ani nawet wyczerpująco nie opisuje, współpracy obu tych kluczowych technologii ze sobą. Można by uznać, że taki stan wynika z koncepcji architektury wielowarstwowej, w której technologie z sąsiednich warstw nie powinny być ściśle zależne od siebie, aby technologie w poszczególnych warstwach można było łatwo zastępować innymi. Jednakże fakt powstawania frameworków zajmujących się integracją JSF z EJB oraz planowane uwzględnienie specyfikacji dotyczącej tej problematyki w wersji 6 platformy Java EE [JEE6] wskazuje na to, że usystematyzowanie współpracy JSF z EJB jest potrzebne.

Celem niniejszego artykułu jest prezentacja istniejących podejść do integracji JSF z EJB. Punktem wyjścia będzie przedstawienie rozwiązań ograniczających się do standardowych specyfikacji Java EE w wersji 5. Następnie przedstawione i porównane pod względem podejścia do wsparcia dla integracji JSF z EJB będą dwa konkurujące ze sobą frameworki: JBoss Seam i Oracle ADF. Wybór akurat tych dwóch frameworków do porównania podyktowany jest faktem, że z popularnych rozwiązań jedynie te dwa oferują konkretne rozwiązania problemu, któremu poświecony jest niniejszy artykuł. Oprócz nich jedynym liczącym się frameworkiem obejmującym wszystkie aspekty aplikacji Java EE jest wspominany już wcześniej Spring. Nie będzie on jednak tu omawiany, gdyż zarówno JSF jak i (w jeszcze większym stopniu) EJB, choć wspierane, mają dla Springa znaczenie marginalne. Co więcej, twórcy Springa, w przeciwieństwie do środowisk związanych z ADF i Seam, nie angażują się w uzupełnienie specyfikacji Java EE o standardowe mechanizmy wiązania warstwy prezentacji z warstwą usług biznesowych, czego szczególnym przypadkiem jest integracja JSF i EJB.

2. Współpraca JSF i EJB w Java EE 5

Omawianie problematyki współpracy JSF z EJB należy rozpocząć od doprecyzowania, że rozważania będą ograniczone do komponentów sesyjnych EJB. Obok komponentów sesyjnych w EJB 3.0 istnieją również komponenty komunikatowe, ale one z założenia nie są bezpośrednio wywoływane przez klientów, w tym np. komponenty warstwy prezentacji, którą zajmuje się JSF. Komponenty komunikatowe są wiązane z kolejką lub tematem w ramach systemu wymiany wiadomości i „wywoływane" asynchronicznie poprzez wysłanie wiadomości do odpowiedniego miejsca przeznaczenia. Z kolei obecne do wersji 2.1 technologii EJB komponenty encyjne, zostały w trakcie prac nad EJB 3.0 „odchudzone" do encji w formie obiektów POJO, którym poświęcona została nowa odrębna specyfikacja Java Persistence API. Encje JPA mogą być wprawdzie wykorzystywane z poziomu JSF bez pośrednictwa sesyjnych EJB, ale zastosowanie tzw. sesyjnej fasady jest powszechnym rozwiązaniem (o statusie wzorca projektowego), choćby dlatego, że rozwiązuje problem obsługi transakcji, co jest mocną stroną technologii EJB.

Jak już wspomniano wcześniej, Java EE 5 nie poświęca zbyt wiele miejsca współpracy JSF z EJB. JSF jest frameworkiem webowym, oferującym rozwiązania w zakresie widoku i kontrolera z wzorca projektowego MVC, a pomijającym kwestię modelu, przy założeniu współpracy z modelami implementowanymi w różnych technologiach. Z kolei specyfikacja EJB, jako technologii dla usług biznesowych, przewiduje różne typy klientów, nie traktując JSF w szczególny sposób. Należy przyznać, że oficjalny przewodnik po platformie w wersji 5 [JBC+08] pokazuje przykłady klientów aplikacyjnych oraz WWW, które korzystają z EJB. W tym uwzględnione jest uzyskanie referencji na sesyjny komponent EJB z poziomu zarządzanego komponentu JSF (ang. managed bean). W oparciu o ten mechanizm można stworzyć aplikację, w której warstwa prezentacji w JSF wykorzystuje usługi biznesowe w EJB. Narzuca się jednak pytanie, czy to wszystko czego powinni oczekiwać twórcy aplikacji Java EE.

Aby zilustrować podejścia do współpracy JSF z EJB, najpierw ograniczając się do rozwiązań z Java EE 5, a następnie korzystając z porównywanych frameworków JBoss Seam i Oracle ADF, rozważymy przykłady pochodzące z aplikacji dla sklepu internetowego. Wśród komponentów logiki biznesowej takiej aplikacji będą znajdowały się komponent koszyka zakupów i komponent zwracający listę dostępnych produktów. Wśród stron-widoków aplikacji będzie strona umożliwiająca przeglądanie listy produktów, dodawanie elementów do koszyka, podgląd zawartości koszyka i usuwanie z niego elementów.

Poniżej przedstawiono kod interfejsu biznesowego (zdalnego) bezstanowego sesyjnego komponentu EJB, umożliwiającego odczyt produktów oraz szkielet jego klasy:

```
@Remote
public interface ItemsCollection {
public List<Item> getAllItems();
```

```
public Item getItem(String id);
}

@Stateless
public class ItemsCollectionBean implements ItemsCollection
{
   ...
}
```

Aby skorzystać z powyższego komponentu EJB z poziomu JSF należy w komponencie zarządzanym JSF (typowo będzie to backing bean związany z którąś ze stron widoków) uzyskać referencję do niego. Najlepiej skorzystać w tym celu z mechanizmu wstrzykiwania zależności, a następnie utworzyć metodę (właściwość) pośredniczącą w wywołaniu metody komponentu EJB, co ilustruje poniższy przykład:

```
public class Home {
   ...
   @EJB
   private ItemsCollection items;
   ...
   public ItemsCollection getItemsCollection() {
      return items;
   }
   ...
}
```

W przypadku gdy wykorzystywany serwer nie wspiera wstrzykiwania zależności do zarządzanych komponentów JSF, można uciec się do klasycznego sposobu wyszukiwania współpracujących komponentów, polegającego na jawnym wyszukiwaniu ich poprzez interfejs JNDI:

```
public class Home {
   ...
   private ItemsCollection items;
   ...
   public ItemsCollection getItemsCollection() {
      ItemsCollection items = null;
      try {
         InitialContext ic = new InitialContext();
         items = (ItemsCollection)
         ic.lookup("JavaEE-Model-ItemsCollection#...ItemsCollection");
      } catch (NamingException e) {
         System.out.println("Error" + e.getMessage());
      }
      return items;
   }
   ...
}
```

Na stronie JSF, z którą związany jest komponent backing bean będący instancją przedstawionej wyżej klasy *Home*, można odwołać się do listy produktów udostępnianej faktycznie przez sesyjny EJB za pomocą poniższego wyrażenia EL (*homeBean* to nazwa komponentu backing bean):

```
#{homeBean.items.itemsCollection}
```

Rozważmy teraz komponent EJB reprezentujący koszyk zakupów. Tym razem będzie to komponent stanowy:

```
@Remote
public interface Cart {
  public List<Item> getCartItems();
  public void addItem(Item item);
  public void removeItem(String id);
  public double getCartTotal();
}

@Stateful
public class CartBean implements Cart
{
```

```
    ...
}
```

Podobnie jak w przypadku komponentu bezstanowego, aby odwołać się do komponentu stanowego na stronie JSF, należy skorzystać z pośrednictwa komponentu zarządzanego. Różnica jest taka, że w tym wypadku programista musi zadbać o przechowanie referencji do stanowego EJB między kolejnymi wywołaniami strony. Do tego celu należy wykorzystać sesję HTTP, albo nadając komponentowi zarządzanemu zasięg sesji, albo w przypadku komponentu zarządzanego o zasięgu żądania jawnie umieścić referencję do EJB w zasięgu sesji, co ilustruje poniższy fragment kodu:

```
public class Home {
   private Cart cart;
   ...
   public Cart getCart() {
      FacesContext context = FacesContext.getCurrentInstance();
      Cart userCart = (Cart)context.getExternalContext()
                           .getSessionMap().get("userCart");
      if ( userCart == null) {
         try {
            InitialContext ic = new InitialContext();
            // JNDI lookup for remote Cart
            cart = (Cart)ic
                              .lookup("JavaEE-Model-Cart#....Cart");
         } catch (NamingException e) {
            System.out.println("Error." + e.getMessage());
         }
      context.getExternalContext()
                  .getSessionMap().put("userCart", cart);
      }
      return userCart;
   }
   ...
}
```

Powyższe przykłady pokazują oczywiście, że do współpracy JSF z EJB nie są wymagane rozwiązania wykraczające poza standard Java EE 5. Niestety nie sposób nie zauważyć, że na programistę spada obowiązek „sklejenia ze sobą" tych technologii poprzez mało odkrywczy, powtarzalny kod umieszczany w komponentach backing beans. Do tego w przypadku stanowych sesyjnych EJB, programista musi zadbać o przechowanie referencji do EJB w warstwie prezentacji, mając do dyspozycji właściwie jedyne sensowne rozwiązanie jakim jest przechowanie referencji w sesji HTTP.

3. JBoss Seam jako framework integrujący JSF i EJB

JBoss Seam to framework oparty na platformie Java EE, który integruje jej poszczególne technologie w celu dostarczenia spójnych, kompleksowych rozwiązań dla wytwarzania aplikacji wielowarstwowych [Seam]. Głównym założeniem Seam jest podjęcie tematu złączenia technologii składowych Java EE, m.in. EJB oraz JSF. Już sama nazwa Seam (po polsku: szew) wskazuje na to, iż framework opisuje w szczególności warstwę pośredniczącą pomiędzy nimi. JBoss Seam bazuje na głównych innowacjach wprowadzonych w wersji 5 platformy Java EE, takich jak używanie adnotacji, wstrzykiwanie zależności, uproszczenie plików konfiguracyjnych XML na rzecz konfiguracji poprzez wyjątki i innych. Technologia ta stara się wykorzystać największe zalety JSF i EJB oraz rozszerza je m.in. poprzez ingerencje w cykl życia stron JSF, czy też rozszerzenia języka wyrażeń EL. Seam ma za zadanie m.in. umożliwienie pominięcie żmudnego pisania dużej ilości kodu pośredniczącego pomiędzy warstwami aplikacji, którego do tej pory nie dało się uniknąć.

Zasadnicze cechy Seam to model komponentowy oraz konteksty. W założeniu, JBoss Seam definiuje model komponentowy, który ma stanowić pełną logikę biznesową tworzonej aplikacji. Komponenty Seam mogą być stanowe lub bezstanowe i osadzone mogą być w różnych kontekstach aplikacji, rozszerzających listę standardowo dostępnych w Java EE zasięgów żądania, sesji i aplikacji o kontekst biznesowy, definiowany w osobnych plikach konfiguracyjnych, czy też

kontekst konwersacji, który może zachowywać ciągłość wielu różnych interakcji z użytkownikiem w obrębie pojedynczej sesji. Ponadto, Seam nie narzuca sztywnego podziału na komponenty należące do logiki biznesowej, czy do warstwy prezentacji. Dzięki temu, programiści mają większą swobodę w tworzeniu architektury swojego systemu, niż w wypadku technologii z góry narzucających pewne ścieżki. Nic nie stoi jednak na przeszkodzie, aby kontynuować korzystanie z dobrze rozpoznanych modeli rozdziału na warstwy, a jedynie korzystać z dobrodziejstw Seama celem zmniejszenia złożoności samego kodu.

Aby zilustrować sposób wiązania EJB z JSF rozważmy sposób implementacji koszyka zakupów jako stanowego sesyjnego EJB będącego jednocześnie komponentem Seama:

```
@Stateful
@Name("cart")
@Scope(ScopeType.SESSION)
public class CartBean implements Serializable, Cart {

   @DataModel
   private List<Item> cartList = new ArrayList<Item>();

   @In(required = false)
   private Item item;

   private double total = 0;

   public Double getCartTotal() {
      return total;
   }

   public CartBean() {
   }

   public void addItem() {
      cartList.add(item);
      total += item.getPrice();
   }

   public void removeItem() {
      cartList.remove(item);
      total -= item.getPrice();
   }

   @Remove
   public void destroy() {
   }

}
```

W definicji komponentu zwracają uwagę przede wszystkim dwie dodatkowe adnotacje (oprócz *@Stateful*) poprzedzające definicję klasy. Pierwsza z nich nadaje komponentowi nazwę, czyniąc z niego komponent Seama. Druga przypisuje komponentowi zasięg, umieszczając go w jednym z wielu oferowanych przez Seam kontekstów (w tym wypadku jest to zasięg sesji).

Do tak zdefiniowanego komponentu Seam można odwołać się bezpośrednio z poziomu strony JSF za pomocą języka wyrażeń EL, np.:

```
<h:commandlink value="Dodaj" action="#{cart.addItem}"/>
```

4. Oracle ADF jako framework integrujący JSF i EJB

Oracle Application Development Framework (ADF) [Ora09a] to framework dostarczany przez jednego z największych potentatów rynku IT na świecie. Oparty jest on o platformę Java EE i oferuje wsparcie na przestrzeni pełnego cyklu życia oprogramowania, w tym celu ściśle współpracując ze zintegrowanym środowiskiem programistycznym (IDE) JDeveloper. Głównym celem ADF jest kompleksowe podejście do problemu wytwarzania złożonych projektów

programistycznych bazujących na dużej ilości, zróżnicowanych źródeł danych, poprzez zapewnienie architektury zorientowanej na możliwość powtórnego użycia. To ostatnie stwierdzenie, poza możliwością powtórnego użycia stworzonych już elementów projektu, wiąże się ze ścisłą współpracą z IDE dostarczonym przez Oracle, które oferuje często deklaratywne i wizualne metody programowania w postaci dużej liczby kompleksowych kreatorów i wszechobecnej techniki drag-and-drop. Techniki te pozwalają programiście pomijać powtarzalne czynności związane z początkiem tworzenia nowych komponentów i minimalizują potrzebę pisania kodu źródłowego w ogóle.

Korzenie Oracle ADF sięgają roku 1999, kiedy to pojawiło się pierwsze użycie komponentów JBO, później BC4J (ang. Business Components for Java), które ewoluowały w obecne ADF Business Components stanowiące główną technologię implementacji logiki biznesowej w najnowszej wersji ADF – 11g [Ora09b]. Aktualna koncentracja ADF na komponentach biznesowych jest szczególnie wyraźna w porównaniu z poprzednią wersją frameworka – 10g, w której ścieżki wykorzystujące ADF Business Components i EJB były przedstawiane jako równie istotne, zorientowane na dwie różne klasy programistów: posiadających doświadczenia z narzędziami 4GL Oracle'a [Ora06b] i programistów Java EE [Ora06a]. Kluczowe obecnie elementy Oracle ADF, oprócz ADF Business Components, to ADF Model odpowiadający za udostępnienie usług biznesowych warstwie prezentacji w jednolitej formie, niezależnie od wybranej technologii ich implementacji, oraz ADF Faces – bogata biblioteka komponentów JSF do budowy interfejsu użytkownika w aplikacjach webowych.

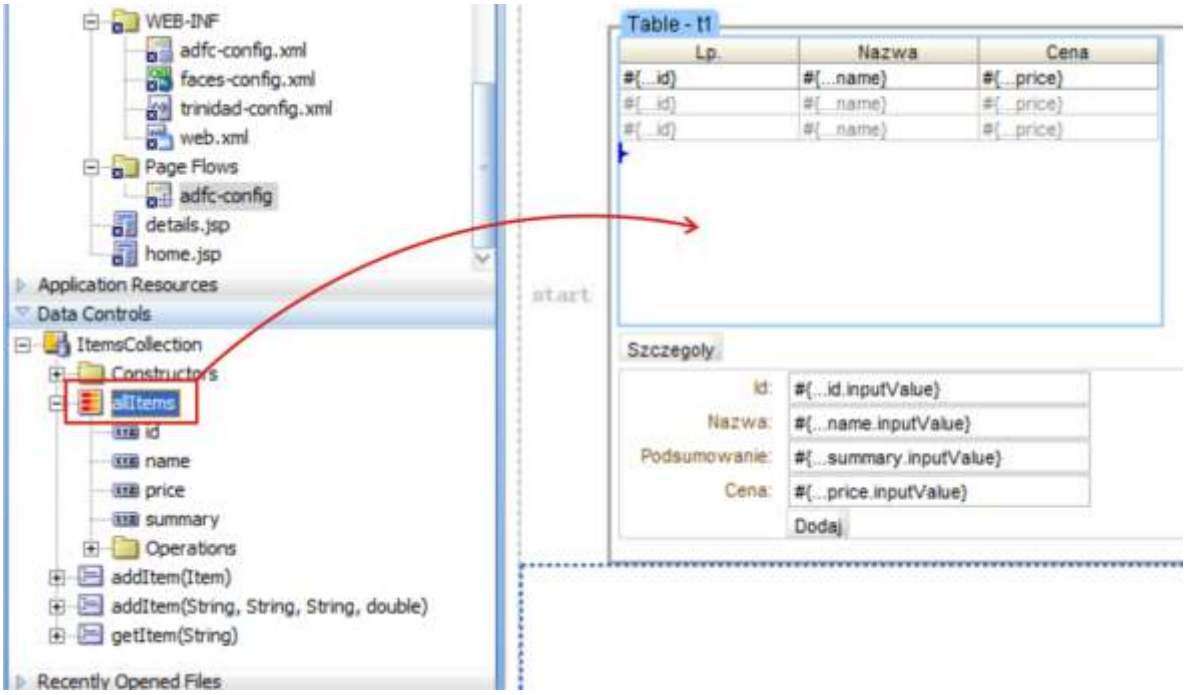
Aby zilustrować sposób wiązania JSF z EJB wrócimy do przykładu bezstanowego sesyjnego EJB udostępniającego listę produktów. Nie będziemy pokazywali obsługi w ADF stanowych sesyjnych EJB (których przykładem jest w rozważanej przykładowej aplikacji koszyk zakupów), gdyż ADF aktualnie nie oferuje dodatkowego wsparcia dla komponentów stanowych, przez co do ich obsługi należałoby wykorzystać techniki przechowywania referencji na komponent w sesji zilustrowane już wcześniej przy omawianiu standardowych mechanizmów Java EE.

Punktem wyjścia do wykorzystania komponentu EJB na stronie JSF w oparciu o deklaratywne wiązanie ADF jest utworzenie kontrolki danych reprezentującej komponent EJB za pomocą prostego kreatora w środowisku JDeveloper. Poniżej przedstawiono treść pliku XML (*ItemsCollection.xml*) z definicją kontrolki danych odpowiadającej komponentowi EJB udostępniającemu informacje o produktach. (Definicje utworzonych kontrolek danych są zawarte w pielęgnowanym przez JDeveloper pliku *DataControls.dcx*.)

```
<JavaBean ... id="ItemsCollection" Package="mordaka.bartosz.pm.adf.model"
      BeanClass="mordaka.bartosz.pm.adf.model.ItemsCollection"
      isJavaBased="true">
   <AccessorAttribute id="allItems" IsCollection="true"
      BeanClass="mordaka.bartosz.pm.adf.model.helper.Item"
      CollectionBeanClass="UpdateableCollection"/>
   <MethodAccessor ...
        BeanClass="mordaka.bartosz.pm.adf.model.helper.Item"
        id="getItem" ReturnNodeName="Item">
      <ParameterInfo id="id" Type="java.lang.String"
                                  isStructured="false"/>
   </MethodAccessor>
   ...
   <MethodAccessor IsCollection="false" Type="void" id="addItem"
        ReturnNodeName="Return">
      <ParameterInfo id="id" Type="java.lang.String"
                                  isStructured="false"/>
      ...
   </MethodAccessor>
</JavaBean>
```

Kolekcję danych udostępnianą przez kontrolkę danych można powiązać deklaratywnie z komponentem interfejsu graficznego (w tym wypadku opartego o komponenty ADF Faces – stanowiącej część ADF biblioteki komponentów JSF) korzystając w środowisku JDeveloper z

techniki drag-and-drop, co ilustruje Rys. 1. W szczególności można w ten sposób zlecić utworzenie nowego elementu wizualnego na stronie, automatycznie powiązanego z kontrolką danych.



Rys. 1. Tworzenie tabeli z danymi korzystając z panelu kontrolek danych

W odpowiedzi na powiązanie zrealizowane przez twórcę aplikacji techniką drag-and-drop, JDeveloper automatycznie utworzy bądź uaktualni pliki XML-owe składające się na aplikację ADF. Poniżej przedstawiona została treść jednego z nich – pliku definicji strony, zawierającego wszystkie powiązania do usług biznesowych dla jednej konkretnej strony. (Ponadto, pielęgnowany jest plik *DataBindings.cpx*, który zawiera definicje kontekstu połączeń, czyli łączy wszystkie pliki definicji stron ze stronami-widokami aplikacji. Plik ten zawiera również odniesienie do kontrolek danych, z którymi powiązane są połączenia zdefiniowane w aplikacji.)

```
<pageDefinition xmlns="http://xmlns.oracle.com/adfm/uimodel"
      version="11.1.1.54.7" id="homePageDef"
      Package="mordaka.bartosz.pm.adf.view.pageDefs">
   <parameters/>
   <executables>
      <variableIterator id="variables">
         ...
         <variable Type="java.lang.String"
                           Name="Item_summary" IsQueriable="false"/>
         ...
      </variableIterator>
      <methodIterator Binds="Item.result"
                DataControl="ItemsCollection"
         id="ItemIterator" ... />
         ...
   </executables>
  <bindings>
   ...
   <attributeValues IterBinding="variables" id="name">
      <AttrNames>
         <Item Value="Item_name"/>
      </AttrNames>
   </attributeValues>
   ...
   <methodAction id="addItem" RequiresUpdateModel="true"
                Action="invokeMethod"
        MethodName="addItem" IsViewObjectMethod="false"
```

```
        DataControl="ItemsCollection"
        InstanceName="ItemsCollection.dataProvider">
      <NamedData NDName="id" NDValue="${bindings.Item_id}"
                 NDType="java.lang.String"/>
      ...
    </methodAction>
    ...
    </bindings>
  </pageDefinition>
```

Samo powiązanie komponentu interfejsu użytkownika na stronie JSF z modelem ADF realizowane jest poprzez język wyrażeń (charakterystyczne jest odwołanie do obiektu *bindings* reprezentującego wszystkie powiązania strony z modelem):

```
<af:table value="#{bindings.allItems.collectionModel}" ... >
```

5. Porównanie ADF i Seam w zakresie integracji JSF z EJB

5.1. Sposób łączenia JSF z EJB

W JBoss Seam mechanizmem integrującym EJB z JSF jest model nazwanych komponentów osadzanych w dobrze zdefiniowanych kontekstach. Mechanizm ten pozwala na luźnie połączenie komponentów składowych aplikacji (ang. loose-coupling), pozostawiając decyzje o sformalizowaniu struktury architektom aplikacji. Znane z JSF obiekty backing beans, przestają mieć tutaj rację bytu w związku z rozszerzeniem języka wyrażeń EL, który pozwala na bezpośrednie odwoływanie się do komponentów Seam obecnych w danym kontekście. Seam pozwala na ograniczenie złożoności warstw prezentacji i logiki biznesowej, przy jednoczesnym umożliwieniu zróżnicowanej złożoności warstwy pośredniej – od zupełnie luźnych połączeń, gdzie różne strony JSF, żądają różnych usług komponentów Seam, po stosowanie dowolnej liczby warstw pośrednich zaczerpniętych z innych wzorców projektowych. Możliwości te zaś dostępne są z nastawieniem na zmniejszoną ilość kodu pośredniczącego m.in. dzięki mechanizmowi dwukierunkowego wstrzykiwania zależności (ang. bijection). JBoss Seam oferuje wsparcie zarówno dla komponentów stanowych jak i bezstanowych sesyjnych EJB, co więcej propagując użycie tych pierwszych a tym samym dementując plotki o domniemanej ich śmierci w świecie rynku produkcyjnego IT z powodu ostatecznie słabego wsparcia dla skalowalności.

Oracle ADF proponuje zupełnie odmienne podejście do tematu integracji EJB z JSF. Jako, że jest to technologia bardzo kompleksowa, z założenia wspierająca szereg technologii będących substytutami tych przytoczonych, współpraca EJB z JSF zostaje zamknięta, jako jedna z wielu ścieżek pokrywanych przez zunifikowany mechanizm współpracy warstwy logiki biznesowej z warstwą prezentacji. Prezentuje on technikę automatycznego tworzenia ujednoliconych interfejsów biznesowych w postaci kontrolek danych (ang. data controls). Zintegrowane środowisko JDeveloper, będące nieodłącznym elementem Oracle ADF udostępnia funkcje automatycznego generowania tych interfejsów z klas logiki biznesowej implementowanych w różnych technologiach, w tym lansowanych przez Oracle ADF Business Components, a także EJB i innych. Bliżej warstwy prezentacji działa zaś mechanizm połączeń danych, który, także w jednolity sposób, odwołuje się do kontrolek danych celem bezpośredniego pobierania wartości własności obiektów biznesowych czy też wywoływania ich metod. W takim razie podobnie jak w Seam, mamy tutaj do czynienia z mechanizmem wprowadzającym pewne abstrakcje pomiędzy EJB i JSF, które pozwalają na spójne przekazywania między nimi odpowiednich danych. W odróżnieniu jednak od tego poprzedniego, brak w ADF wsparcia dla stanowych sesyjnych komponentów EJB. Producent mocno uwypukla architekturę bezstanową ściśle związaną ze światem relacyjnych baz danych i to właśnie ją chce zaoferować programistom. W tym świetle, komponenty EJB są raczej traktowane jako jedno z wielu źródeł danych, dla bardziej złożonych, kompleksowych systemów.

5.2. Poziom skomplikowania

Mechanizmy łączenia EJB i JSF w Seam i ADF znacząco różnią się poziomem skomplikowania. W Seam ilość kodu niezbędna do implementacji podstawowych funkcji jest minimalna i ogranicza się do dodania do istniejącego kodu kilku adnotacji. Dzięki temu, cała aplikacja zyskuje na prostocie z racji braku w niej komponentów pośredniczących backing beans. Proste aplikacje w Seam można pisać z ograniczeniem ilości zarówno kodu, jak i klas składowych. W miarę rozrastania się aplikacji w Seam, poziom skomplikowania jednak nieznacznie rośnie. Co prawda, średnia ilość kodu źródłowego przypadająca na daną implementowaną funkcję jest raczej stała, to jednak prawdopodobnie w wypadku większych projektów, niezbędne będzie zaproponowanie dodatkowych warstw pośrednich. Ta decyzja, a zarazem docelowa architektura, pozostawiona jest do stworzenia przez programistów, a celem takich działań, byłoby uproszczenie struktury aplikacji celem ułatwienia jej dalszego rozwoju i utrzymania.

W ADF sprawa ponownie ma się zupełnie inaczej. Tutaj nawet bardzo proste aplikacje będą składać się ze znacznej liczby klas oraz plików XML stanowiących metadane. Ręczne pisanie tych ostatnich, w świetle ich skomplikowania i objętości, zdaje się być niemożliwe do zrealizowania w praktycznych zastosowaniach. Na szczęście, w zdecydowanej większości wypadków, nie będzie takiej potrzeby, z racji faktu, iż generowane są one przez narzędzie JDeveloper. Średnia ilość kodu na funkcjonalność, podobnie jak w Seam, raczej powinna być stała, natomiast w miarę rozrastania się aplikacji w ADF nie ma mowy o zwiększeniu komplikacji. Wyjściowa struktura, która co prawda ma dość dużą złożoność, jest na tyle dobrze przygotowana, iż nie będzie ulegała komplikacji w miarę przybywania nowych klas i nowych zależności pomiędzy widokami.

5.3. Wygoda korzystania, środowiska programistyczne, serwery, licencje

JBoss Seam oraz Web Beans to technologie otwarte i darmowe rozpowszechniane na otwartej licencji LGPL. Aplikacje stworzone z ich wykorzystaniem mogą być uruchamiane na wszystkich kompatybilnych z Java EE serwerach bez dodatkowych licencji. Omawiając Seam, należy jednak wspomnieć o środowisku programistycznym JBoss Developer Studio, które jest produktem komercyjnym i do jego pobrania i korzystania wymagana jest odpowiednia licencja.

Oracle ADF jest technologią komercyjną. Bez dodatkowej licencji, aplikacje z niej korzystające można uruchamiać tylko na serwerze aplikacyjnym pochodzącym od Oracle, który, jak łatwo się domyślić, sam w sobie jest płatny. Niemniej jednak, w odróżnieniu od produktów JBoss, jeden z najważniejszych elementów ADF, czyli środowisko JDeveloper jest produktem, który nie wymaga dodatkowych licencji nawet dla potrzeb komercyjnych.

5.4. Literatura i dokumentacja

JBoss Seam jest technologią dobrze udokumentowaną. Oficjalny przewodnik jasno przedstawia funkcjonalność frameworka, podając do tego wiele przykładów. Język jest przystępny, jego obszerność nie przytłacza, a jednocześnie pokrywa on wszystkie aspekty technologii. Poza tym, od roku 2005, kiedy ukazała się pierwsza wersja Seam, ukazało się kilka pozycji literaturowych (np. [All09], [YuHe07]), które omawiają zarówno podstawowe jak i zaawansowane tematy dotyczące korzystania z Seam. Pozwala to zainteresowanym programistom dość niskim nakładem czasu zaznajomić się z podstawowymi koncepcjami Seam i szybko przystąpić do działań praktycznych.

W przypadku Oracle ADF mamy do czynienia z technologią udokumentowaną obszernie, aczkolwiek słabo. Należy wspomnieć fakt, iż przez pierwsze lata dostępności ADF na rynku, brak było jakiejkolwiek dokumentacji jemu poświęconej. Obecnie literatura dotycząca ADF sprowadza się do dokumentacji pochodzącej od producenta. Ta zaś jest ściśle powiązana z rozwijaniem oprogramowania korzystającego z ADF w JDeveloperze, co utrudnia zrozumienie istoty technologii i zamyka jej wszechstronność poprzez ograniczanie do jednego tylko przedstawionego rozwiązania. Dokumentacja ADF (szczególnie w wersji 11g) jest silnie nastawiona na propagowanie

komponentów ADF Business Components jako logiki biznesowej dla aplikacji i trudno w niej znaleźć przykłady dla innych alternatyw. Oficjalny przewodnik sprawia wrażenie tendencyjnego, struktura jego rozdziałów i ich tytuły nie ułatwiają wybiórczego znalezienia w niej potrzebnych informacji. W ogólności, poziom udokumentowania Oracle ADF, nie tylko w przypadku dokumentacji producenta, pozostawia wiele do życzenia. Pomimo mnogości przykładów i małych przewodników, brak jest pozycji, która podsumowywałaby w przystępny sposób najważniejsze cechy i sposoby użytkowania ADF. To zaś sprawia, iż pomimo dokumentacja zdaje się dotykać większości aspektów zaimplementowanych w ADF, to jednak ma ona taką formę, iż programiści chcący rozpocząć pracę z tą technologią potrzebują dużego nakładu czasu. Zaznajomienie się zaś z większością koncepcji ADF oraz JDevelopera, może być jednak gwarantem dla sprawnego i szybkiego tworzenia oprogramowania zgodnie z wytycznymi podanymi przez Oracle.

5.5. Standaryzacja

Najnowsza wersja platformy Java EE, czyli wersja szósta [JEE6], która jeszcze w 2009 roku powinna ujrzeć światło dzienne, pośród nowych standardów przyniesie m.in. technologię Web Beans, opisaną w JSR-299: Java Context and Dependency Injection [JSR299]. Specyfikacja ta została zaproponowana i jest rozwijana głównie przez Gavina Kinga, pracownika JBoss, który jest jednocześnie pomysłodawcą i twórcą technologii JBoss Seam. W związku z tym, siłą rzeczy, wiele idei z Seam znajduje swoje odzwierciedlenie w Web Beans, który jednakże czerpie również z Google Guice [Guice] (bezpieczeństwo typowania w procesie wstrzykiwania zależności) oraz Spring [Spring].

Oracle podjął próbę zaszczepienia kluczowej dla ADF koncepcji deklaratywnego wiązania warstwy prezentacji z usługami biznesowymi w oparciu o abstrakcję w postaci kontrolek danych w ramach specyfikacji JSR-227: A Standard Data Binding & Data Access Facility for J2EE [JSR227]. Jak sugeruje numer JSR, prace nad specyfikacją zainicjowaną przez Oracle rozpoczęły się wcześniej niż nad specyfikacją związaną z Seam. Mimo to, prace te ciągle trwają i na razie nie ma informacji na temat uwzględnienia tej specyfikacji w nadchodzących wersjach Java EE. Być może wynika to po części z faktu, że we wstępnym głosowaniu specyfikacja nie uzyskała jednoznacznej aprobaty. IBM i BEA głosowały przeciw, jako powód podając zbyt duży ich zdaniem zakres specyfikacji.

6. Podsumowanie

JSF i EJB to kluczowe technologie platformy Java EE. Pierwsza stanowi framework do implementacji warstwy prezentacji, a druga jest „oficjalną" technologią do implementacji logiki biznesowej. Niestety wersja 5 platformy Java EE w bardzo niewielkim stopniu zajmuje się kwestiami współpracy JSF z EJB, przez co programiści muszą tracić czas na tworzenie powtarzalnego kodu „sklejającego" JSF z komponentami EJB, typowo w obrębie komponentów backing beans.

Niewątpliwą lukę w specyfikacji Java EE 5 starają się wypełnić frameworki autorstwa środowisk zewnętrznych w stosunku do firmy Sun: JBoss Seam i Oracle ADF. Frameworki te, mimo że pokrywają się funkcjonalnością, różnią się znacząco od siebie, w szczególności prezentując skrajnie odmienne podejścia do integracji JSF z EJB. Różnice te z pewnością w dużym stopniu wynikają z genezy obu frameworków. Seam w pełni opiera się na technologiach Java EE i jest zorientowany na programistów doświadczonych w tej technologii, skupiając się na wykorzystaniu jak najlepszych dostępnych rozwiązań i wypełniając zidentyfikowane luki. Z kolei ADF powstał z myślą o programistach korzystających wcześniej z narzędzi 4GL oferowanych wcześniej przez Oracle, a także o rozwoju własnego oprogramowania wcześniej tworzonego w tych narzędziach.

JBoss Seam integruje JSF z EJB w oparciu o rozwinięte wstrzykiwanie zależności i konteksty w których dostępne są obiekty. Cechuje się przejrzystością kodu, przez co nie wymaga IDE to tworzenia aplikacji. Z pewnością jego zaletą jest fakt, że koncepcje w nim obecne zostaną podniesione do rangi standardu w Java EE 6, a także to, że już obecnie Seam wykorzystuje JSF w połączeniu z technologią Facelets zamiast JSP, co będzie usankcjonowane w Java EE 6. Wadą Seama, choć jego twórcy mają inne zdanie, jest zatarcie granic między warstwą prezentacji, a warstwą logiki biznesowej. Czy faktycznie jest to wada czy zaleta pokażą zapewne dopiero doświadczenia z projektów opartych o Seam.

Oracle ADF stawia zdecydowany nacisk na deklaratywno-wizualne wiązanie stron-widoków JSF z usługami biznesowymi. Ułatwia to tworzenie typowych aplikacji, ale pogarsza czytelność kodu, ze względu na automatycznie generowane liczne pliki XML. Z punktu widzenia programistów Java EE zniechęcające może być to, że o ile w ADF 10g technologia EJB była przedstawiana jako pełnoprawna alternatywna dla będących firmowym rozwiązaniem Oracle ADF Business Components, to w ADF 11g stos technologii zawierający rozwiązania specyficzne dla Oracle jest zdecydowanie preferowany. Ponadto, pomysły zawarte z ADF ciągle nie mogą przebić się do standardów Java EE, a do tego ADF wolniej niż Seam nadąża za nowinkami w technologiach Java EE. ADF jest więc atrakcyjnym rozwiązaniem dla programistów migrujących z wcześniejszych narzędzi Oracle i/lub oczekujących kompletnego rozwiązania zorientowanego na produkty Oracle.

Podsumowując różnice między Seam i ADF, nie można nie wspomnieć o tym, że nawet podejście do licencji jest w przypadku tych frameworków odmienne. Seam jest darmowy, ale oferowane przez jego twórców środowisko IDE jest płatne. Oracle zastosował strategię dokładnie odwrotną: środowisko JDeveloper jest bezpłatne, a licencja wymagana jest zarówno w przypadku chęci użycia ADF w aplikacji, jak i później do jej uruchamiania na serwerze produkcyjnym. Niedogodność tę w pewnym stopniu zmniejsza fakt, że licencje na serwery aplikacji Oracle obejmują również licencję ADF.

Bibliografia

[All09] Allen D., Seam in Action, Manning, 2009

[Guice] Google Guice, http://code.google.com/p/google-guice/

[HaMa05] Harrop R., Machacek J., Pro Spring: Spring and EJB, JavaWorld.com, 2/14/05

[JEE] Java EE at a Glance, http://java.sun.com/javaee/index.jsp

[JEE6] Java EE 6 Technologies, http://java.sun.com/javaee/technologies/javaee6.jsp

[JBC+08] Jendrock E., Ball J., Carson D., Evans I., Fordin S., Haase K., The Java EE 5 Tutorial For Sun Java System Application Server 9.1, 2008

[JoHo04] Johnson R., Hoeller J., Expert One-on-One J2EE without EJB, Wrox, 2004

[JSR227] JSR-227: A Standard Data Binding & Data Access Facility for J2EE, http://jcp.org

[JSR299] JSR-299: Java Context and Dependency Injection, http://jcp.org

[Ora06a] Oracle Application Development Framework Developer's Guide 10g Release 3 (10.1.3), 2006

[Ora06b] Oracle Application Development Framework Developer's Guide For Forms/4GL Developers 10g Release 3 (10.1.3.0), 2006

[Ora09a] Oracle Application Development Framework Overview, An Oracle White Paper, 2009

[Ora09b] Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework 11g Release 1 (11.1.1), 2009

[Seam] Seam Framework, http://seamframework.org

[Spring] Spring, http://www.springsource.org/

[YuHe07] Yuan M., Heute T., JBoss Seam: Simplicity and Power Beyond Java EE, Prentice Hall, 2007