XIII Konferencja PLOUG
Kościelisko
Październik 2007

JHeadstart 10.1.3 – nowa jakość tworzenia aplikacji Java EE

Mikołaj Morzy
Instytut Informatyki Politechniki Poznańskiej

Mikolaj.Morzy@put.poznan.pl

Abstrakt. Architektura wielowarstwowa szybko staje się *de facto* standardem tworzenia nowoczesnych aplikacji. Elastyczność, otwartość, produktywność, oraz uproszczona pielęgnacja aplikacji czynią architekturę wielowarstwową bardzo atrakcyjnym wyborem projektowym. Niestety, budowanie w pełni wielowarstwowych aplikacji wymaga od osób projektujących i implementujących aplikacje bardzo wysokich kwalifikacji oraz znajomości różnych narzędzi, interfejsów i protokołów wykorzystywanych w ramach poszczególnych warstw aplikacji. Stąd, wiele przedsiębiorstw i organizacji rezygnuje z inwestowania w tworzenie aplikacji wielowarstwowych z obawy przed niepowodzeniem takich projektów.
JHeadstart to nowe narzędzie, którego celem jest maksymalne uproszczenie procesu tworzenia aplikacji wielowarstwowych. JHeadstart ściśle integruje się ze środowiskiem programistycznym JDeveloper i umożliwia deklaratywne projektowanie aplikacji wielowarstwowej w technologii ADF w oparciu o wygenerowany wcześniej model danych oraz bogaty zbiór preferencji i metadanych. Wykorzystanie narzędzia JHeadstart pozwala na błyskawiczne stworzenie aplikacji wielowarstwowej bez konieczności napisania nawet jednej linijki kodu. Sam proces tworzenia aplikacji jest podobny do procesu tworzenia aplikacji w architekturze klient-serwer przy użyciu narzędzia Oracle Forms. JHeadstart stanowi atrakcyjną alternatywę dla organizacji rozważających migrację z Oracle Forms na platformę wielowarstwową.
W niniejszym artykule przedstawiono szczegółowo narzędzie JHeadstart. Opisano poszczególne elementy narzędzia: edytor graficzny, generator aplikacji, wtyczkę do Oracle Designer umożliwiającą generowanie aplikacji wielowarstwowych bezpośrednio na podstawie definicji z repozytorium Oracle Designer, oraz zestaw komponentów JHeadstart Runtime rozszerzających funkcjonalność ADF Framework. Rozważono konsekwencje wykorzystania JHeadstart jako narzędzia migracji aplikacji Oracle Forms na platformę Java EE, podkreślając zalety i wady takiej decyzji. Pokrótce omówiono także kwestie licencjonowania narzędzia oraz kwestie zgodności JHeadstart z innymi narzędziami Oracle.

Informacja o autorze. Dr inż. Mikołaj Morzy jest adiunktem w Instytucie Informatyki Politechniki Poznańskiej. Wcześniej pracował na Uniwersytecie w Muenster (Niemcy) i Loyola University w Nowym Orleanie (Stany Zjednoczone). Jego zainteresowania naukowe koncentrują się przede wszystkim na tematyce eksploracji danych, jest on autorem ponad czterdziestu publikacji dotyczących tej tematyki. Drugą dziedziną zainteresowań i głównym tematem działalności dydaktycznej Mikołaja Morzego są technologie i architektury szkieletowe dla aplikacji internetowych i rozproszonych.

1. Wprowadzenie

W ostatnich latach jesteśmy świadkami gwałtownej migracji aplikacji przygotowanych w architekturze klient-serwer na platformę wielowarstwową. Wśród platform wielowarstwowych prym wiodą platforma Java Enterprise Edition (Java EE), wspierana przez firmy Sun, Oracle i IBM, oraz platforma .NET, wspierana przez firmę Microsoft. Popularność aplikacji wielowarstwowych jest ściśle związana z zaletami takiej architektury. W porównaniu z aplikacjami klient-serwer aplikacje wielowarstwowe charakteryzują się większą elastycznością, łatwością pielęgnacji kodu, rozszerzalnością, oraz umożliwiają ścisły podział obowiązków pomiędzy osoby odpowiedzialne za tworzenie kolejnych warstw oprogramowania. Z drugiej strony, tworzenie aplikacji wielowarstwowych wymaga od osób tworzących oprogramowanie dużo większych kompetencji oraz dogłębnej znajomości co najmniej kilku zazębiających się ze sobą technologii. W powszechnym przekonaniu tworzenie aplikacji wielowarstwowych jest dużo trudniejsze niż programowanie tradycyjnych systemów i w znacznie mniejszym stopniu jest wspierane przez narzędzia programistyczne, szczególnie przez narzędzia do modelowania i automatycznej generacji aplikacji.

W niniejszym artykule przedstawiono Oracle JHeadstart 10.1.3 – jedno z pierwszych profesjonalnych narzędzi do modelowania i deklaratywnego tworzenia aplikacji wielowarstwowej w technologii Java EE poprzez automatyczną generację aplikacji na podstawie metadanych. Artykuł ma następującą strukturę. W rozdziale 2 przedstawiono krótką charakterystykę wielowarstwowej architektury aplikacji. Rozdział 3 przedstawia JHeadstart – proces generowania aplikacji, strukturę metadanych, oraz dodatkowe elementy JHeadstart Runtime. W rozdziale 4 zaprezentowano możliwości wykorzystania JHeadstart do generowania aplikacji wielowarstwowych na podstawie metadanych przechowywanych w repozytorium Oracle Designer oraz przedyskutowano użycie JHeadstart do migracji aplikacji Oracle Forms na platformę Java EE. Rozdział 5 to informacje na temat licencjonowania oprogramowania, wsparcia dla osób tworzących aplikacje przy użyciu JHeadstart oraz kwestii różnic między poszczególnymi wersjami. Artykuł kończy rozdział 6 zawierający króciutkie podsumowanie.

2. Architektura wielowarstwowa aplikacji

Najważniejszą cechą odróżniającą architekturę wielowarstwową (ang. *multi-tier architecture*) od tradycyjnej architektury klient-serwer jest wyraźne odseparowanie w strukturze aplikacji warstw realizujących odmienne zadania. Na Rysunku 1 przedstawiono schematycznie warstwy składające się na architekturę wielowarstwową.

Rys. 1. Architektura wielowarstwowa aplikacji

Na najniższym poziomie znajduje się warstwa zasobów (ang. *resource tier*), w której umieszczone są serwery baz danych, serwery plików oraz dostawcy innych zasobów konsumowanych przez aplikację (np. kolejki komunikatów, usługi Web Services, itp.) Na najwyższym poziomie znajduje się warstwa klienta (ang. *client tier*), za pomocą której użytkownicy zgłaszają swoje żą-

dania. Najczęściej zakłada się, że klient spełnia minimalne wymagania dotyczące implementowanych interfejsów. Przykładowo, w kontekście aplikacji internetowych często pojawia się pojęcie „cienkiego" klienta (ang. *thin client*). Jedynym wymaganiem odnośnie cienkiego klienta jest aby klient był w stanie komunikować się za pomocą protokołu HTTP i potrafił konsumować odpowiedzi przesyłane w formacie HTML lub XML. Pomiędzy warstwą zasobów i warstwą klienta znajduje się warstwa środkowa (ang. *middle tier*), która często jest logicznie podzielona na warstwę prezentacji (ang. *presentation tier*) i warstwę biznesową (ang. *business tier*). Warstwa biznesowa to komponenty odpowiedzialne za takie zadania, jak: komunikacja z warstwą zasobów, implementacja logiki biznesowej, implementacja logiki transakcyjnej, oraz wymuszanie złożonych reguł biznesowych. W warstwie tej najczęściej spotykamy takie rozwiązania jak komponenty sesyjne i encyjne EJB, komponenty biznesowe ADF, czy kod własny. Warstwa prezentacji jest odpowiedzialna za obsługę żądań przychodzących z warstwy klienta, wywoływanie metod udostępnionych przez warstwę biznesową, oraz formatowanie wyników zwróconych przez metody warstwy biznesowej do postaci akceptowalnej przez klienta.

Taka architektura jest bez wątpienia bardziej skomplikowana niż tradycyjna architektura klient-serwer. Wprowadzenie zestawu komunikujących się ze sobą warstw nakłada narzut architektoniczny związany przede wszystkim z przekazywaniem parametrów i wyników pomiędzy poszczególnymi warstwami. Warto także zauważyć, że na styku między kolejnymi warstwami wykorzystywane są zupełnie inne interfejsy. Przykładowo, warstwa biznesowa i warstwa zasobów komunikują się ze sobą najczęściej za pomocą języka SQL, warstwa klienta komunikuje się z warstwą prezentacji za pomocą protokołu HTTP i języka HTML (lub podobnego), natomiast sposób komunikacji między warstwą prezentacji i warstwą biznesową zależy od wykorzystywanej platformy. W powszechnym przekonaniu koszt związany z większą komplikacją wewnętrznej architektury i struktury aplikacji z nawiązką rekompensują zalety architektury wielowarstwowej. Najważniejsze zalety to:

- **niezależność od interfejsu klienta**: sposób komunikacji z klientem jest istotny tylko i wyłącznie z punktu widzenia warstwy prezentacji, logika biznesowa i sposób dostępu do danych jest niezależny od klienta, aplikacja może obsługiwać jednocześnie heterogenicznych klientów, w stosunku do których sformułowano tylko podstawowe wymagania (np. obecność przeglądarki internetowej obsługującej język JavaScript),
- **niezależność od warstwy zasobów**: analogicznie, dostęp do warstwy danych jest wykonywany przez specjalizowane komponenty warstwy biznesowej i w przypadku zmiany dostawcy danych tylko niewielka część aplikacji musi zostać uaktualniona,
- **centralizacja aplikacji**: logika biznesowa i logika prezentacji są realizowane niezależnie od klienta na specjalizowanej jednostce zwanej serwerem aplikacji, w konsekwencji wszelkie modyfikacje oprogramowania wykonywane są tylko i wyłącznie w jednym miejscu, ułatwiając pielęgnację kodu, zarządzanie bezpieczeństwem aplikacji, czy ochronę własności intelektualnej.

Powyższe czynniki w dużej mierze przesądziły o powodzeniu architektury wielowarstwowej jako podstawowej architektury tworzenia nowoczesnych aplikacji w ostatnich latach. Niestety, skomplikowanie architektury i wielość technologii wchodzących w skład aplikacji wielowarstwowej powodują, że przygotowanie aplikacji wielowarstwowej od podstaw wymaga bardzo wysokich kompetencji i biegłości w obsłudze tak różnych technologii jak: protokół HTTP, język Java, język SQL, czy praca z kontenerem EJB. Stąd, wiele organizacji przejawia daleko posuniętą wstrzemięźliwość przed migracją do architektury wielowarstwowej. Dodatkowo, przez długi czas wsparcie architektury wielowarstwowej z poziomu narzędzi CASE było niewystarczające, a dostępne środowiska programistyczne wspierały tworzenie aplikacji wielowarstwowych tylko w wąskich wycinkach, np. automatyzując proces rejestracji serwletu w deskryptorach wdrożenia. Niedawno na rynku pojawiły się pierwsze narzędzia umożliwiające tworzenie aplikacji wielowarstwowych zgodnie z zasadami inżynierii oprogramowania, rozpo-

czynając od etapu modelowania konceptualnego, a na deklaratywnej specyfikacji aplikacji kończąc. Przykładem takiego narzędzia jest JHeadstart, opisany w kolejnym rozdziale.

3. Tworzenie aplikacji przy użyciu JHeadstart

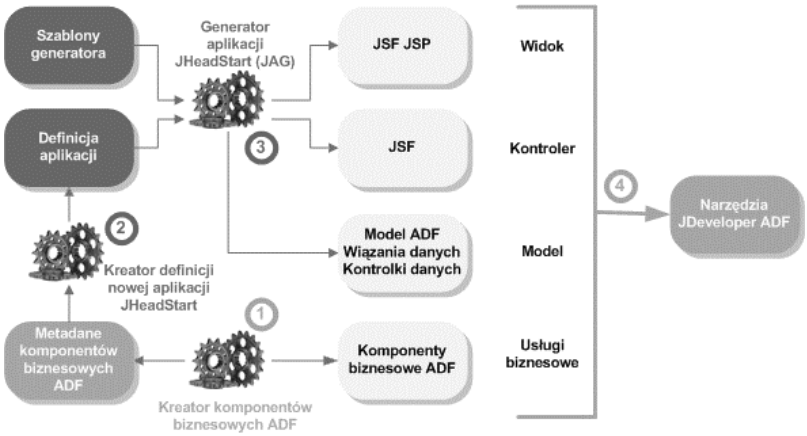
3.1. Wprowadzenie do JHeadstart

Oracle JHeadstart to rozszerzenie funkcjonalności środowiska programistycznego JDeveloper, umożliwiające szybką pracę nad aplikacją wielowarstwową. Zwiększenie prędkości i efektywności pracy zostało osiągnięte poprzez wykorzystanie potężnego generatora, który automatycznie generuje gotową aplikację w technologii Oracle Application Development Framework (ADF) na podstawie bogatego zestawu metadanych. Metadane są specyfikowane przez użytkownika poprzez prosty edytor własności, przypominający edytor własności dobrze znany z takich narzędzi jak Oracle Forms czy Oracle Reports. Metadane są następnie tłumaczone na zestaw komponentów JSF lub ADF i łączone do postaci wynikowych dokumentów prezentowanych użytkownikowi. JHeadstart automatyzuje proces wypełniania dokumentu wynikowego kontrolkami JSF, wiązanie kontrolek z komponentami zarządzanymi, wiązanie komponentów z modelem ADF BC, obsługę nawigacji między dokumentami oraz zachowanie transakcyjne poszczególnych dokumentów. Poniższa lista stanowi podsumowanie zalet wykorzystania JHeadstart jako narzędzia tworzenia aplikacji wielowarstwowych:

- zwiększona produktywność: stworzenie w pełni funkcjonalnego prototypu może być kwestią minut pracy pod warunkiem dysponowania modelem danych,
- naturalne przejście na platformę Java EE: brak konieczności generowania nawet pojedynczej linii kodu, deklaracja metadanych odbywa się poprzez mechanizm podobny do tworzenia aplikacji przy użyciu Oracle Forms, możliwość łatwej adaptacji osób przyzwyczajonych do środowiska Oracle Developer,
- spójny wygląd aplikacji: automatyczna generacja całości aplikacji zapewnia 100% zgodność wyglądu i zachowania dokumentów wynikowych,
- integracja z narzędziami Oracle Designer i Oracle Forms: możliwość generacji aplikacji na podstawie metadanych z repozytorium Oracle Designer oraz możliwość inżynierii zwrotnej aplikacji Oracle Forms do postaci aplikacji wielowarstwowych,
- niezależność od konkretnej technologii: JHeadstart generuje całą aplikację wynikową na podstawie zbioru metadanych zapisanych w postaci plików XML, konkretna realizacja aplikacji może być zmieniana na poziomie generatora (przykładowo, w wersji 10.1.2 JHeadstart główną implementacją były technologie Struts i UIX, podczas gdy wersja 10.1.3 bazuje na Java Server Faces i ADF).

Na Rysunku 2 przedstawiono schematycznie proces tworzenia aplikacji wielowarstwowej przy wykorzystaniu narzędzia JHeadstart. W kroku pierwszym kreator komponentów biznesowych ADF generuje usługę ADF składającą się z komponentów biznesowych i metadanych. Komponenty biznesowe ADF stanowią warstwę modelu, na którym będzie tworzona faktyczna aplikacja. Krok generowania komponentów biznesowych ADF jest całkowicie niezależny od JHeadstart. W drugim kroku kreator definicji nowej aplikacji JHeadstart tworzy szkielet gotowej do użycia aplikacji wielowarstwowej na podstawie obiektów wchodzących w skład modelu komponentów biznesowych. Definicja nowej aplikacji jest w rzeczywistości zbiorem plików XML zawierających wszystkie informacje o tworzonej aplikacji. Na podstawie tej definicji w kroku trzecim generator aplikacji JHeadstart (JAG) generuje gotową aplikację wielowarstwową, tworząc odpowiednio: zbiór dokumentów JSP składających się na warstwę widoku, plik konfiguracyjny `faces-config.xml` do sterowania kontrolerem JSF, oraz wiązania danych i kontrolki danych składające

się na warstwę modelu wynikowej aplikacji. Wygenerowana aplikacja jest już w pełni funkcjonalna. Dalsza praca z narzędziem JHeadstart polega na iteracyjnych zmianach metadanych i wielokrotnym uruchamianiu generatora JAG w celu weryfikacji uzyskanej aplikacji. Rzecz jasna, w idealnej sytuacji nie powinno być konieczne wprowadzanie żadnych zmian w wygenerowanej aplikacji. W praktyce jednak wiele elementów aplikacji musi być oprogramowane ręcznie. Dotyczy to np. reguł logiki biznesowej implementowanej na poziomie modelu lub widoku. Takie zmiany można wprowadzić do kodu źródłowego aplikacji przygotowanego przez generator JAG wykorzystując udogodnienia oferowane przez zintegrowane środowisko programistyczne JDeveloper, np. zmiany w wyglądzie stron można wygodnie wprowadzić przy pomocy wizualnego edytora ADF.



Rys. 2. Tworzenie aplikacji JHeadStart od podstaw

W tym miejscu należy wyraźnie podkreślić, że generator aplikacji JHeadstart nie jest generatorem w tradycyjnym rozumieniu tego słowa. Automatyczna generacja kodu przywodzi na myśl negatywne konotacje, ponieważ bardzo często kod wygenerowany automatycznie jest niespójny, trudny do analizy i ręcznej modyfikacji, czy nadmiernie skomplikowany. Wady te nie występują w przypadku wykorzystania generatora JHeadstart, ponieważ narzędzie to nie generuje kodu źródłowego Java. W przeciwieństwie do innych narzędzi produktem działania generatora jest zbiór plików konfiguracyjnych wiążących ze sobą poszczególne części aplikacji (np. plik konfiguracyjny `faces-config.xml`) oraz strony JSP. W kolejnych podrozdziałach przedstawiamy podstawowe elementy wchodzące w skład narzędzia JHeadstart.

3.2. Definicja aplikacji

Najważniejszym elementem metadanych wykorzystywanych przez generator JAG jest definicja aplikacji. Jest to plik XML zawierający definicję wszystkich źródeł danych, listę stylów układu poszczególnych stron wynikowych, listę kontrolek w generowanym interfejsie użytkownika, specyfikację możliwych zapytań, jakie można wydawać za pomocą formularzy na stronach wynikowych oraz specyfikację zachowania transakcyjnego stron wynikowych. Definicja aplikacji jest złożonym plikiem, ale wykorzystując kreator definicji nowej aplikacji, stanowiący część JHeadstart, można bardzo szybko stworzyć szkielet definicji, który posłuży do wygenerowania pierwszego prototypu. Dalsza praca z JHeadstart sprowadza się właściwie do iteracyjnego nanoszenia zmian do pliku definicji aplikacji i generowania na tej podstawie aplikacji wynikowej tak długo, aż uzyskany efekt usatysfakcjonuje osobę programującą. Zmiany w definicji aplikacji mogą być wprowadzane ręcznie, ale rzecz jasna znacznie szybciej, prościej i wygodniej jest posłużyć się edytorem definicji aplikacji. Narzędzie to wyglądem i funkcjonalnością bardzo przypomina edytor palety właściwości dobrze znany z Oracle Forms czy JDeveloper. Jest to okno zawierające listę

elementów aplikacji pogrupowanych logicznie w kategorie i podkategorie. Każda podkategoria stanowi listę właściwości, które można dowolnie zmieniać. Dla każdej właściwości dostępny jest również wyczerpujący opis pozwalający na stwierdzenie, jaki wpływ na wygląd i zachowanie wygenerowanej aplikacji będzie miało ustawienie cechy danej właściwości. W zależności od typu danych właściwości do ustawienia właściwości edytor wykorzystuje właściwą kontrolkę: pole tekstowe, pole wyboru lub listę rozwijaną. Edytor definicji aplikacji pozwala również na kopiowanie całych fragmentów (elementów lub grup własności) czy ich przenoszenie za pomocą techniki „przeciągnij i upuść". Dodatkowo, edytor zapewnia pracę w dwóch trybach: podstawowym i eksperckim. W trybie podstawowym wiele zaawansowanych właściwości jest ukrytych i przyjmują wartości domyślne, w trybie eksperckim osoba tworząca oprogramowanie ma dostęp do wszystkich właściwości.

Definicja aplikacji jest hierarchicznie zorganizowanym zbiorem własności. Poszczególne gałęzie hierarchii własności mogą reprezentować następujące elementy:

- **usługa (ang. *service*)**: jest to korzeń hierarchii, reprezentuje logicznie powiązany fragment aplikacji i jest oparty na pojedynczym module aplikacyjnym ADF BC, właściwości usługi odpowiadają globalnym ustawieniom aplikacji i można tu, między innymi, zdefiniować globalny format daty, lokalizacje plików, ustawienia polityk bezpieczeństwa, czy skonfigurować ustawienia narodowe.
- **grupa (ang. *group*)**: reprezentuje pojedynczą kolekcję danych i definiuje sposób pracy z tą kolekcją (w terminologii Oracle Forms grupa opowiada mniej więcej blokowi danych), w przypadku ADF BC grupa wykorzystuje obiekt widoku i określa dozwolone operacje (wstawianie, modyfikacja, usuwanie rekordów), dodatkowo, właściwości grupy szczegółowo określają szablon rozłożenia kontrolek na stronie (układ formularzowy, układ tabelaryczny, drzewo, lista wyboru z formularzem, układ tabelaryczny z formularzem szczegółowym, itp.) Wewnątrz grupy mogą występować grupy zagnieżdżone (ang. *detail groups*), za pomocą których można bardzo łatwo tworzyć formularze typu „*master-detail*".
- **element (ang. *item*)**: reprezentuje pojedynczą daną, może być związany z danymi źródłowymi (ang. *databound item*) – w takim przypadku reprezentuje najczęściej wartość jednego atrybutu, albo może być niezwiązany (ang. *unbound item*) – wówczas reprezentuje element interfejsu użytkownika, taki jak przycisk, odnośnik, czy inna kontrolka. Wśród własności znajdują się, między innymi, własności do określania wyglądu (pole tekstowe, lista rozwijana, pole memo, przycisk radiowy, wykres, pasek postępu, itp.), własności do określania funkcjonalności (dozwolone wstawianie, modyfikacja), wartość domyślna, czy wskazanie zależności względem innego elementu.
- **kontener (ang. *container*)**: kontener jest zbiorem regionów elementów (ang. *item region*), które z kolei są logicznymi grupami elementów, które mogą posiadać dodatkową nazwę i być wyświetlone np. w postaci zakładek. Oprócz grup elementów kontener może również, na analogicznych zasadach, zawierać region grup (ang. *group region*), dla których określone zostaną wspólne zasady wyświetlania zawartości.
- **lista wartości (ang. list of values, LOV)**: JHeadstart pozwala na wypełnianie pól wartościami sczytanymi z listy wartości w sposób bardzo dobrze znany użytkownikom Oracle Forms. Lista wartości jest implementowana w postaci niezależnego okienka wypełnionego danymi, dla których można stosować dodatkowe filtry oraz wykorzystywać mechanizm „*use LOV for validation*", polegający z grubsza na wymuszeniu poprawności wprowadzanych danych poprzez ich konfrontację z zawartością listy wartości.
- **domena (ang. *domain*)**: statyczny lub dynamiczny zbiór wartości które mogą być użyte jako lista poprawnych wartości atrybutu, w aplikacji wynikowej może być wyświetlana jako lista rozwijana, pole wyboru, pole radiowe, w przypadku domen dynamicznych można jako własność domeny wskazać na kolekcję danych służącą do dostarczania wartości domeny.

3.3. Generator aplikacji JHeadstart

Generator aplikacji JHeadstart generuje, na podstawie definicji aplikacji opisanej w poprzednim punkcie, aplikację wynikową. Cały proces generacji jest w całości sterowany szablonami a wykorzystywana technologia to *Velocity* (http://jakarta.apache.org/velocity). *Velocity* to narzędzie umożliwiające dynamiczne odwołania do obiektów Java z poziomu szablonu wypełnionego dowolną statyczną treścią. Odwołania są specyfikowane przy użyciu prostego języka szablonów VTL (ang. *Velocity Template Language*). Szablony *Velocity* służą do generowania zarówno wynikowych stron JSP, jak i do automatycznego generowania zawartości pliku konfiguracyjnego `faces-config.xml`. JHeadstart posiada bogatą bazę szablonów umożliwiających generowanie, na podstawie definicji aplikacji, dużej liczby różnych typów stron JSP: formularzy typu „*master-detail*", drzew, odnośników nawigacyjnych, list wartości, słowem wszelkiego rodzaju stron JSP składających się na aplikację wynikową. Osoba tworząca aplikację JHeadstart może dowolnie modyfikować każdy szablon, wpływając tym samym na sposób generacji i format aplikacji wynikowej. Poniżej nieco bardziej szczegółowo opisano rodzaje obiektów generowanych automatycznie przez generator JAG:

- **dokumenty JSF JSP**: generator JAG generuje dokumenty JSP, czyli strony JSP przygotowane w formacie XML, dzięki czemu dokumenty takie są łatwo parsowane przez edytory wizualne i ich późniejsza ręczna modyfikacja nastręcza mniej problemów.
- **`faces-config.xml`:** najważniejszymi elementami wygenerowanego pliku konfiguracyjnego `faces-config.xml` są reguły nawigacyjne i definicje komponentów zarządzanych, przy czym jako komponenty zarządzane występują tylko i wyłącznie specjalizowane klasy Java dostarczane wraz z narzędziem JHeadstart, w ten sposób wszystkie zaawansowane elementy interfejsu użytkownika (pola do wydawania zaawansowanych zapytań, listy wartości, formularze w kształcie drzewa, wielowierszowe wstawianie) zależą od klas dostarczanych przez JHeadstart i są umieszczane w aplikacji wynikowej jako komponenty zarządzane. Dodatkowo, umożliwia to użytkownikom specjalizację klas i dostosowywanie powtarzalnych elementów interfejsu do swoich potrzeb.
- **pliki PageDef**: plik PageDef przechowuje informacje o klasach wykonywalnych dla poszczególnych kontrolek danych i zawiera definicję wiązania elementów strony z kontrolką danych, dla każdej generowanej strony plik PageDef jest tworzony automatycznie.
- **zbiory zasobów**: zbiór zasobów to plik zawierający listę etykiet tekstowych przystosowanych do wyświetlania dokumentu w wybranym języku, przygotowując kilka zbiorów zasobów można w łatwy sposób automatycznie generować różne wersje językowe tej samej aplikacji.

Jak już wcześniej wspomniano, tworzenie aplikacji JHeadstart jest procesem iteracyjnym, w trakcie którego generator JAG wielokrotnie wygeneruje aplikację wynikową na podstawie przygotowanej definicji aplikacji. Ponieważ w każdej aplikacji część logiki, specyficzna dla danej aplikacji, musi być wprowadzona ręcznie, pojawia się problem nadpisywania zmian wprowadzonych ręcznie przez nowe wersje generowane przez JAG. Rodzi się zatem pytanie, w jaki sposób można zabezpieczyć jakiś element aplikacji przed nadpisaniem podczas automatycznej generacji aplikacji. JHeadstart oferuje trzy podejścia do tego problemu. Najprostsze rozwiązanie polega na zaznaczeniu na poziomie metadanych (czyli definicji aplikacji), że wybrany fragment ma nie podlegać generacji. Właściwość taka może być ustawiona na poziomie całej usługi (nie zostaną wówczas wygenerowane menu, pliki zasobów i plik `faces-config.xml`), oraz na poziomie grupy (nie zostaną wówczas wygenerowane definicja strony JSP, wiązania komponentów zarządzanych oraz reguły nawigacyjne dla danej strony). Warto także pamiętać o tym, że jakiekolwiek zmiany wprowadzone ręcznie do pliku konfiguracyjnego `faces-config.xml` nigdy nie zostaną nadpisane w procesie generacji. Jeśli ręcznych zmian jest bardzo niedużo, można je udokumentować i wprowadzić ponownie po zakończonej generacji. Rozwiązanie takie jest stosowalne tylko i wyłącznie

wtedy, gdy lista zmian wprowadzonych do wygenerowanej aplikacji jest bardzo krótka. Wreszcie, kompromisem między wcześniejszymi dwoma rozwiązaniami jest modyfikacja szablonu wykorzystywanego do generowania aplikacji. Ponieważ każdy element interfejsu użytkownika jest tworzony na podstawie osobnego szablonu, można utrwalić zmiany wprowadzane ręcznie bezpośrednio na poziomie szablonu i wykorzystywać zmieniony szablon. Zaletą takiego podejścia jest fakt, że umożliwia ono ponowne zaaplikowanie ręcznie wprowadzonych zmian na dowolnym poziomie aplikacji oraz pozwala na zastosowanie tych samych zmian w wielu różnych miejscach generacji, gdziekolwiek wskażemy zmodyfikowany szablon. Wadą tego rozwiązania jest konieczność opanowania języka VTL w celu poprawnego modyfikowania szablonów *Velocity*.

3.4. JHeadstart Runtime

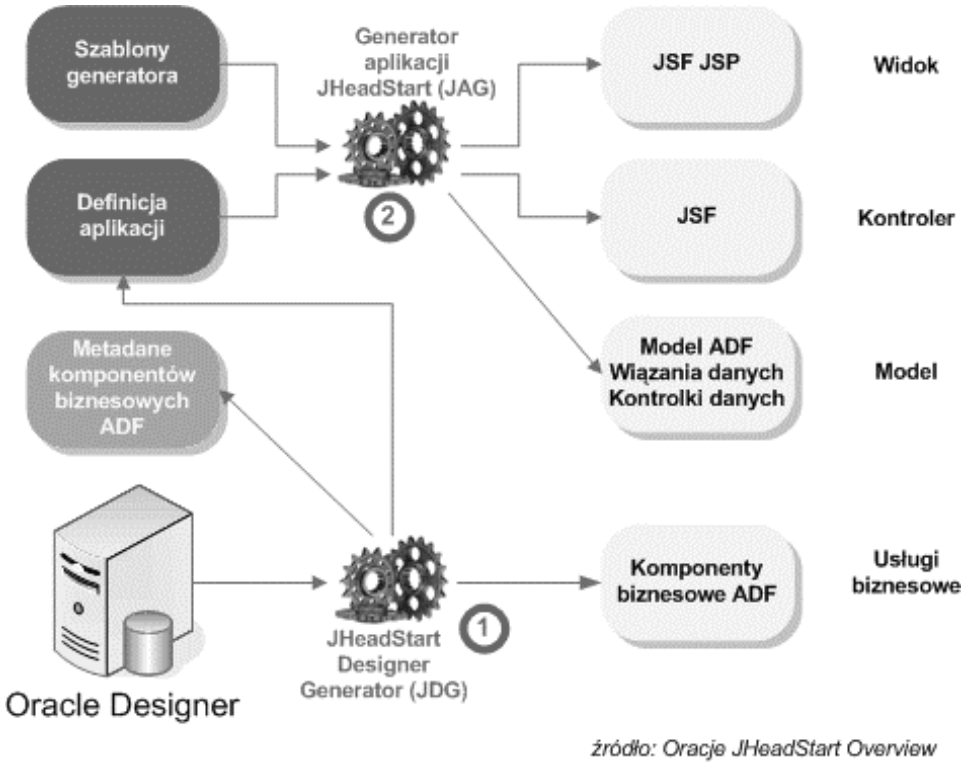
JHeadstart Runtime to zbiór klas języka Java (są dystrybuowane wraz z ich kodem źródłowym), biblioteka JavaScript oraz arkusz stylistyczny CSS. Te trzy komponenty wspólnie składają się na wygląd i funkcjonalność aplikacji wynikowej. Arkusz stylistyczny CSS jest wykorzystywany do zapewnienia spójnego wyglądu wszystkich dokumentów wynikowych aplikacji. Zmiany wprowadzone w arkuszu wpłyną bezpośrednio na wygląd generowanych dokumentów, co ogromnie ułatwia personalizację tworzonych aplikacji. Biblioteka JavaScript zawiera pomocnicze funkcje, ułatwiające reagowanie na zdarzenia zachodzące w trakcie działania aplikacji oraz kopiujące zachowania znane użytkownikom z Oracle Forms (np. funkcja `alertForChanges()`) wyświetla komunikat o niezatwierdzonych zmianach, analogiczny do okna powiadomienia wyświetlanego przez Oracle Forms. Wreszcie pakiety klas języka Java dostarczają klas użytkowych rozszerzających funkcjonalność klas wykorzystywanych przez ADF. Przykładowo, klasa `JhsPageLifecycle` umożliwia reagowanie na zdarzenia utworzenia wiersza, zatwierdzenia lub wycofania transakcji, czy usunięcie wiersza, klasa `JhsListOfValues` implementuje automatyczną walidację pól na podstawie listy wartości, a klasa `ReportingUtils` pozwala na wyświetlanie komunikatów o błędach w sposób przyjazny użytkownikowi.

4. JHeadstart i migracja z Oracle Forms na platformę Java EE

3.1. Generowanie aplikacji Java EE z repozytorium Oracle Designer

Jedną z najważniejszych cech narzędzia JHeadstart jest możliwość wygenerowania w pełni funkcjonalnej aplikacji wielowarstwowej na podstawie modelu przechowywanego w repozytorium Oracle Designer. Nie powinno to budzić niczyjego zdziwienia, krótki rzut okiem na metadane wykorzystywane przez JHeadstart uświadamia, że struktura metadanych, ich podział na kategorie, a nawet niektóre nazwy właściwości zostały skopiowane z metadanych Oracle Designer. Podstawowa różnica polega na niezgodności formatów, w których przechowywane są metadane. Oracle Designer przechowuje swoje metadane w repozytorium, na które składa się zbiór tabel w relacyjnej bazie danych. Z kolei JHeadstart jako repozytorium metadanych wykorzystuje zbiór plików XML. Konwersją pomiędzy tymi formatami metadanych zajmuje się narzędzie zwane JHeadstart Designer Generator (JDG).

W wyniku działania JHeadstart Designer Generator metadane z repozytorium Oracle Designer zostają przetransformowane do postaci plików XML, a definicje tabel, kluczy i perspektyw służą generatorowi JDG do automatycznego zbudowania komponentów biznesowych ADF, które zostaną wykorzystane w warstwie modelu aplikacji wielowarstwowej. Proces transformacji składa się z dwóch kroków przedstawionych na Rysunku 2.



Rys. 3. Tworzenie aplikacji JHeadstart w oparciu o zawartość repozytorium Oracle Designer

W pierwszym kroku generator JDG odczytuje metadane z repozytorium Oracle Designer, i na ich podstawie tworzy kolejno: komponenty biznesowe ADF, metadane dla komponentów biznesowych ADF, oraz definicję aplikacji JHeadstart. Komponenty biznesowe ADF są budowane w następujący sposób: definicje tabel zostają zaimplementowane w postaci obiektów ADF BC typu Entity, definicje elementów modułów z Oracle Designer zostają zaimplementowane w postaci obiektów ADF BC typu View, zaś same definicje modułów zostają zaimplementowane w postaci modułów aplikacji ADF BC. W drugim kroku generator JAG pobiera stworzoną definicję aplikacji i w oparciu o szablony generatora tworzy pliki warstw widoku, modelu i kontrolera aby zbudować aplikację ADF. Zazwyczaj, pierwszy krok jest wykonywany tylko jeden raz, a dalsza praca nad aplikacją polega na modyfikowaniu metadanych JHeadstart. Jeśli jednak fragmenty metadanych Oracle Designer uległyby zmianie i konieczna stałaby się ponowna generacja komponentów biznesowych ADF BC, wówczas zmiany wprowadzone do definicji grup na poziomie metadanych JHeadstart mogą zostać uchronione przed nadpisaniem przez włączenie cechy *JDG Protected* na poziomie własności grupy.

Istnieje także możliwość wykorzystania narzędzia JHeadstart do migracji formularzy napisanych w technologii Oracle Forms bez wykorzystania Oracle Designer. Dysponując formularzami przygotowanymi „ręcznie" i planując ich migrację na platformę Java EE należy posłużyć się techniką inżynierii zwrotnej (ang. *reverse engineering*). Oracle Designer posiada narzędzie zwane Design Capture Wizard, umożliwiające przygotowanie metadanych na podstawie istniejącego formularza lub raportu. W rezultacie użycia tego narzędzia do repozytorium Oracle Designer trafiają następujące metadane: definicja modułu, definicje okien, komponenty modułu, informacja o tabelach wykorzystywanych jako źródło danych dla bloku i dla list wartości, informacje o elementach zarówno związanych, jak i niezwiązanych, a także informacje o grupach elementów i argumentach (parametrach) modułu. Wymienione metadane mogą być następnie wykorzystane przez generator JDG do przygotowania szkieletu aplikacji wielowarstwowej.

W tym miejscu należy zwrócić uwagę na to, że JHeadstart Designer Generator nie jest aktualnie częścią JHeadstart w wersji 10.1.3.1. Włączenie generatora JDG jest planowane w drugiej

wersji produkcyjnej, zaś w chwili obecnej generator JDG może być używany z wersjami JHeadstart 10.1.2 i JDeveloper 10.1.2. Z kolei metadane wygenerowane przez JDG z wersji JHeadstart 10.1.2 są bez problemu akceptowane przez generator JAG w wersji JHeadstart 10.1.3.

7.2. Migracja na platformę Java EE

Pytanie o konieczność migracji z technologii Oracle Forms na platformę Java EE tylko z pozoru wydaje się łatwe. W rzeczywistości decyzja taka zależy od wielu czynników, a jej konsekwencje mogą być ogromne i bardzo kosztowne. Poniżej przedstawiono najważniejsze czynniki, które należy rozważyć przed migracją aplikacji na platformę Java EE:

- **Przyzwyczajenia użytkowników**: praca z aplikacją, dla której front-end stanowi przeglądarka, wymaga oswojenia się ze specyficznym interfejsem aplikacji webowej. Migracja aplikacji Oracle Forms na platformę Java EE często wiąże się z koniecznością starannego przygotowania i przyuczenia szerokiej rzeszy użytkowników końcowych aplikacji. W szczególności, aplikacje webowe charakteryzują się mniejszą interaktywnością interfejsu. Jeśli aplikacja wymaga dużej interaktywności, wówczas podczas migracji należy się zastanowić, czy przepisanie aplikacji przy wykorzystaniu komponentów ADF zapewni użytkownikom wymagany stopień interaktywności.
- **Rodzaj aplikacji**: jeśli aplikacja jest bardzo duża, stanowi ważną składową infrastruktury informatycznej organizacji, czy jej działanie zależy od zależności czasowych między poszczególnymi komponentami, wówczas nagłe przejście na platformę Java EE jest ryzykownym wyborem. W takim przypadku warto rozważyć migrację częściową, opisaną poniżej.
- **Zasoby ludzkie**: tworzenie aplikacji wielowarstwowych wymaga zupełnie innych umiejętności niż pisanie aplikacji w architekturze klient-serwer. Dotyczy to zarówno nowych języków programowania, jak i znajomości znacznie większej liczby technologii pomocniczych. Udana migracja z technologii Oracle Forms do platformy Java EE wymaga zatem znaczącej inwestycji w umiejętności i kwalifikacje osób zajmujących się tworzeniem, rozwojem i pielęgnacją aplikacji. W tym punkcie warto sobie więc zadać fundamentalne pytanie: czy migracja na platformę Java EE rzeczywiście jest niezbędna i konieczna?

Sensownym kompromisem między gwałtowną migracją na nową platformę a pozostawieniem aplikacji w starej architekturze jest migracja częściowa. Większość dużych i złożonych systemów informatycznych można logicznie podzielić na funkcjonalne podsystemy, z których każdy posiada swoją własną grupę użytkowników. Stąd, analiza przyzwyczajeń użytkowników, ich zadowolenia z posiadanej aktualnie aplikacji, oraz konsekwencji migracji na platformę Java EE powinna być przeprowadzona niezależnie dla każdej grupy użytkowników. W wyniku takiej analizy może się okazać, że istnieją fragmenty systemu informatycznego które mogą być migrowane niezależnie od pozostałych składników. Taka częściowa migracja posiada bardzo wiele zalet. Po pierwsze, ryzyko związane z przeniesieniem aplikacji na nową platformę gwałtownie spada, ponieważ można rozpocząć migrację od małego niekrytycznego systemu. Po drugie, migracja częściowa pozwala ocenić i przetestować narzędzie wybrane do migracji i zidentyfikować krytyczne punkty migracji. Po trzecie, migracja częściowa daje możliwość zebrania miar oceny migracji (czas, zasoby, zadowolenie użytkowników). Miary oceny można ekstrapolować w celu uzyskania przybliżonych szacunków na temat kosztu migracji większych komponentów systemu informatycznego.

W trakcie migracji aplikacji z technologii Oracle Forms na platformę Java EE bardzo ważną decyzją jest wybór narzędzia, które posłuży do migracji. W portalu Oracle Forms Product Center (http://www.oracle.com/technology/products/forms) utrzymywana jest lista certyfikowanych dostawców oprogramowania do migracji aplikacji. Decydując się na wybór konkretnego narzędzia, należy kierować się dwoma podstawowymi czynnikami: architekturą aplikacji po migracji do platformy Java EE oraz funkcjonalnością oryginalnej aplikacji która podlega

automatycznej migracji. Przykładowo, JHeadstart wykorzystuje jako architekturę wynikową standardy JSF i ADF, które zapewniają elastyczność i otwartość, dodatkowo umożliwiając uruchamianie aplikacji bez konieczności instalowania jakichkolwiek bibliotek i wtyczek po stronie klientów. Inne narzędzia mogą wykorzystywać specjalny aplet Swing do uruchamiania aplikacji po konwersji. W efekcie funkcjonalność i wygląd aplikacji po migracji będą bardzo zbliżone do oryginalnej aplikacji Oracle Forms, jednak użytkownicy końcowi będą musieli instalować w przeglądarce kompatybilną wersję JVM. Narzędzia do konwersji różnią się także ilością automatycznie przenoszonego kodu. JHeadstart nie potrafi przenosić kodu PL/SQL umieszczonego w wyzwalaczach (oczywiście, chodzi o kod logiki biznesowej niezwiązany z funkcjonalnością Oracle Forms). Taki kod musi zostać ręcznie przeniesiony do właściwej warstwy (baza danych, model, kontroler, widok, usługi biznesowe) z wykorzystaniem techniki właściwej dla wybranej warstwy. Z drugiej strony, aplikacja migrowana za pomocą JHeadstart jest rozwijana i modyfikowana w sposób bardzo podobny do pracy z Oracle Forms, co zdecydowanie ułatwia pracę osobom zajmującym się migracją i rozwojem aplikacji. Należy także wyraźnie zaznaczyć, że wybór JHeadstart jako narzędzia do migracji posiada także wady. Jak wcześniej wspomniano, kod PL/SQL umieszczony w wyzwalaczach nie podlega migracji i jego przeniesienie może być czasochłonnym zajęciem. Po drugie, migracja musi się odbywać poprzez repozytorium Oracle Designer, więc przeniesienie ręcznie zbudowanych formularzy wiąże się z koniecznością wykonania inżynierii zwrotnej. Wreszcie, funkcjonalność, zachowanie i wygląd aplikacji wynikowej będą znacząco się różnić od oryginalnej aplikacji Oracle Forms, co może pociągać za sobą konieczność przyzwyczajenia użytkowników końcowych.

8. Licencja, wsparcie i galimatias z wersjami

8.1. Licencja

Oracle JHeadstart 10.1.3.1 jest wersją produkcyjną, podlegającą licencjonowaniu. Koszt pojedynczej licencji deweloperskiej dla nazwanego użytkownika to aktualnie 1750$, przy czym wymagane jest zakupienie minimum trzech takich licencji. Dodatkowo, możliwość pobierania uaktualnień i suplementów jest osobno wyceniona na 15% kosztu pojedynczej licencji rocznie. W ramach tego abonamentu właściciel oprogramowania uzyskuje dostęp do uaktualnień (*point releases*), łatek, poprawek, dodatkowych narzędzi użytkowych, itp. Do celów poznawczych i testowych można pobrać z `otn.oracle.com` wersję ewaluacyjną, jest to w pełni funkcjonalna wersja, za pomocą której jednak nie wolno tworzyć żadnych aplikacji dla środowisk produkcyjnych. Licencje na Oracle JHeadstart są dystrybuowane poprzez Oracle Consulting.

8.2. Wsparcie

Ponieważ Oracle JHeadstart jest zupełnie osobnym produktem stworzonym przez Oracle Consulting, nie można (przynajmniej na razie) uzyskać pomocy i wsparcia przez tradycyjne kanały: Oracle Support Services lub Metalink. Nie oznacza to jednak, że potencjalny użytkownik jest pozostawiony samemu sobie. Bogatym źródłem informacji na temat produktu jest JHeadstart Product Center (http://www.oracle.com/technology/products/jheadstart). Znajduje się tam, między innymi, gotowa do pobrania wersja ewaluacyjna narzędzia, szczegółowy samouczek, liczna i obszerna dokumentacja, a także przykładowe filmy obrazujące wykorzystanie JHeadstart. Odpowiedzi na wiele pytań znajdują się w dokumencie FAQ dostępnym pod adresem http://www.oracle.com/technology/products/jheadstart/files/jheadstart_FAQ.html. Osoby zajmujące się produkcją mogą się także dzielić swoimi doświadczeniami, poszukiwać pomocy oraz zgłaszać propozycje ulepszeń za pomocą internetowego forum poświęconego produktowi JHeadstart (http://forums.oracle.com/forums/forum.jspa?forumID=38). Nieocenionym źródłem wiedzy

o JHeadstart jest także blog internetowy twórców tego narzędzia, dostępny pod adresem http://blogs.oracle.com/jheadstart.

8.3. Wersje

JHeadstart jest produktem nowym i intensywnie rozwijanym. Jak już wcześniej wspomniano, nowa wersja narzędzia przyniesie istotne zmiany. Planując wykorzystanie JHeadstart do tworzenia aplikacji J2EE dobrze jest określić właściwe wersje oprogramowania, aby uniknąć niepotrzebnych konfliktów i komplikacji. Poniższa tabela przedstawia kompatybilność wersji JHeadstart z wersjami środowiska programistycznego Oracle JDeveloper.

Tabela 1. Tabela kompatybilności

Wersja narzędzia JDeveloper	JHeadstart 10.1.3.0	JHeadstart 10.1.3.1
JDeveloper 10.1.3.0 Studio Edition	tak [a]	tak [a,b]
JDeveloper 10.1.3.1 Studio Edition	tak [c]	tak [c]
JDeveloper 10.1.3.2 Studio Edition	nie	tak
JDeveloper 10.1.3.3 Studio Edition	nie	tak

Uwagi:

- W każdym przypadku występuje konflikt między wersją biblioteki `velocity-dep-1.3.jar` dostarczaną przez JDeveloper a wersją biblioteki wykorzystywaną przez JHeadstart. Konieczne jest zatem przemianowanie pliku `velocity-dep-1.3.jar` umieszczonego w katalogu `$JDEV_HOME/jdev/lib` przed użyciem narzędzia JHeadstart.
- a – konieczne jest wyposażenie narzędzia JDeveloper co najmniej w uaktualnienie Service Update 2, którego można dokonań za pomocą opcji **Help | Check for Updates** (przy czym należy w pierwszym kroku zainstalować uaktualnienie Service Update 1, a następnie zainstalować najwyższe dostępne uaktualnienie)
- b – występuje konflikt między wersją biblioteki `xml.jar` dostarczaną przez JDeveloper a wersją biblioteki wykorzystywaną przez JHeadstart. Konieczne jest zamknięcie narzędzia JDeveloper, przemianowanie pliku `$JDEV_HOME/jdev/lib/xml.jar` i przekopiowanie tam pliku `$JDEV_HOME/extensions/oracle.jheadstart.10.1.3/designtime/lib/xml.jar`
- c – w celu uniknięcia znanego błędu związanego z kontrolką do przesyłania plików na serwer należy zainstalować łatkę JDeveloper Patch 5623059

8.4. Co znajdzie się w wersji JHeadstart 10.1.3.2?

Na jesień roku 2007 zapowiadana jest nowa wersja JHeadstart oznaczona numerem 10.1.3.2, która przyniesie wiele istotnych zmian. Bez wątpienia najważniejszą zmianą jest wprowadzenie tzw. komponentów aplikacji wielokrotnego użytku (ang. *reusable application components*). Komponenty takie będą się charakteryzować dużą elastycznością i będą mogły być z powodzeniem wykorzystywane w wielu aplikacjach. Cechy te uzyskano dzięki wprowadzeniu do komponentów następujących elementów:

- identyfikacja i autoryzacja: w nowej wersji domyślnie wykorzystywany jest standard Java Authentication and Authorization Service (JAAS),
- elementy elastyczne (ang. *flex items*): nowe elementy mogą być dodawane do grup w trybie runtime, cechy elementów standardowych mogą być również ustalane w trybie runtime (przykładowo, pole może dynamicznie stać się wymagane lub nieedytowalne),
- tekst wyświetlany na stronie może być tłumaczony i modyfikowany w trybie runtime,

- w warstwie modelu obiekty View Object i Entity Object będą mogły reprezentować dowolne źródło danych, np. usługę sieciową Web Service, dzięki czemu znacznie łatwiejsze stanie się budowanie aplikacji zgodnie z paradygmatem Service Oriented Architecture (SOA),
- możliwość generowania skryptów DDL do tworzenia schematu bazy danych,
- możliwość generowania skryptów SQL do tworzenia ról i nadawania przywilejów,
- możliwość równoległego generowania stron administracyjnych do zarządzania funkcjonowaniem aplikacji.

9. Podsumowanie

Oracle JHeadstart 10.1.3 istotnie stanowi zupełnie nową jakość w tworzeniu aplikacji wielowarstwowych na platformie Java EE. Niezależnie od wyboru konkretnego szkieletu aplikacyjnego (ADF, JSF, Struts) umożliwia błyskawiczne prototypowanie aplikacji, tworzenie skomplikowanych aplikacji bez konieczności żmudnego programowania, automatyzuje ogromną liczbę czynności oraz udostępnia spójny interfejs użytkownika.

Bibliografia

[Dav07] Davelaar S., Oracle JHeadstart 10.1.3.2 New Features
[JhsFAQ] Oracle JHeadstart 10g - Frequently Asked Questions,
[JhsWP] Oracle JHeadstart Overview – An Oracle Technical White Paper, June 2006
[Mue06] Muench S., Jump-Start J2EE Development, Oracle Magazine, Nov/Dec 2006