

XII Konferencja PLOUG
Zakopane
Październik 2006

Oracle Reports 9i/10g, XML Publisher – raportowanie w internecie

Jarosław Gramacki, Artur Gramacki

*Uniwersytet Zielonogórski
Instytut Informatyki i Elektroniki
ul. Podgórna 50, 65-246, Zielona Góra
e-mail: j.gramacki@iie.uz.zgora.pl, a.gramacki@iie.uz.zgora.pl*

Abstrakt

W pracy omówiono najnowsze wersje narzędzi firmy Oracle wspierających tworzenie aplikacji raportujących. Przedstawiono moduł *Reports* z pakietu *Oracle Developer Suite* oraz, w kontekście porównania z nim, nowe rozwiązanie firmy Oracle *XML Publisher*. W części poświęconej pierwszemu rozwiązaniu skupiono się m.in. na tych elementach, które są istotne z punktu widzenia udostępniania raportów na serwerze w architekturze trójwarstwowej J2EE oraz ich uruchamiania z poziomu przeglądarki internetowej. Pokazano również pewne zagadnienia związane z bezpieczeństwem modułu *Reports*. Zwrócono również uwagę na nowy format zapisu raportu (jako pliku JSP) oraz potencjalne możliwości pobierania do raportu również danych zewnętrznych. W części poświęconej *XML Publisher* skupiono się na pokazaniu istotnych z praktycznego punktu widzenia różnic pomiędzy znanym już od dość dawna modułem *Reports* a nowym narzędziem, jakim jest *XML Publisher*.

1. Wprowadzenie

Celem artykułu jest przedstawienie głównych cech dwóch aktualnie dostępnych rozwiązań raportujących firmy Oracle – modułu *Reports* z pakietu *Developer Suite* oraz pakietu *XML Publisher*.

W wielu miejscach przedstawiono również stosunkowo podstawowe zagadnienia związane z budową i obsługą raportów, gdyż dokumentacja firmowa, choć bardzo obszerna, nigdzie nie zawiera niestety krótkiego i przystępnie napisanego wprowadzenia w istotę zagadnienia.

W module *Reports*, począwszy od wersji 9i (która stosunkowo szybko została zastąpiona wersją 10g) w praktyce zrezygnowano z uruchamiania raportów w środowisku klient-serwer na rzecz uruchamiania ich wyłącznie z poziomu przeglądarki internetowej w architekturze trójwarstwowej OC4J (*Oracle Containers for J2EE* – serwer aplikacji J2EE dostarczany przez Oracle). Siłą rzeczy pojawiły się więc zupełnie nowe zagadnienia związane z udostępnianiem raportów na wybranym serwerze aplikacji, sposobami ich wywoływania, parametryzowania czy zabezpieczania. Ważne są również nowe możliwości łączenia w jednym raporcie danych pochodzących ze źródeł nie tylko relacyjnych, ale i innych, jak na przykład plików XML, arkuszy kalkulacyjnych, plików tekstowych, usług Web Services, innych źródeł dostępnych poprzez protokół JDBC (firma Oracle promuje produkt m.in. używając hasła: *Any Data*).

Tworząc dany raport pracujemy w znanym od wersji 6i i jedynie lekko zmodyfikowanym w wersji 9i/10g środowisku projektowym *Reports Builder*. Przy tworzeniu raportów w nowym środowisku można wykorzystywać dwa różne podejścia. W pierwszym z nich tzw. raporty papierowe (ang. *Paper Layout*) tworzone są tak jak w wersji 6i w środowisku projektowym *Reports Builder*, a istotnym novum jest tylko to, że mogą być one uruchamiane w architekturze trójwarstwowej J2EE poprzez przeglądarkę WWW. W drugim podejściu, budując raport, korzysta się wyłącznie z popularnych i znanych technologii internetowych (Java, JSP, XML). Technologie te same w sobie nie są w żaden sposób związane ze wspomnianym wcześniej środowiskiem projektowym, choć w pewnym stopniu są przez to środowisko wspierane; mówimy wówczas, o tzw. raportach webowych (ang. *Web Layout*).

Inne podejście oferuje pakiet *XML Publisher* (w skrócie XMLP). Prosta w obsłudze aplikacja uruchamiana z poziomu przeglądarki WWW umożliwia określenie szkieletu raportu (źródło danych, ewentualne parametry raportu, plik(i) z szablonem raportu). Dalszym krokiem, najbardziej pracochłonnym, jest przygotowanie szablonu raportu w środowisku dowolnego edytora obsługującego format RTF (ang. *Rich Text Format*), np. Microsoft Word. Szablon ten, mówiąc w dużym skrócie, definiuje rozmieszczenie na płaszczyźnie raportu danych pobranych z wybranego źródła danych. Do dyspozycji mamy zbiór kilkunastu obsługiwanych przez pakiet XMLP „firmowych” znaczników wywodzących się bezpośrednio od znaczników XSL [Tra00] oraz dużą liczbę obsługiwanych przez system znaczników formatujących XSL-FO [Tra02] interpretowanych w wersji oryginalnej. Znaczniki te mogą być wstawiane do dokumentu RTF bezpośrednio w postaci tekstowej (ang. *Basic RTF Method*) lub pośrednio poprzez pola formularzy (ang. *Form Field Method*). Ograniczeniem pierwszej metody jest niemożność posługiwania się oryginalną wersją znaczników XSL (które przydatne są szczególnie przy wstawianiu wykresów do raportu) oraz znaczników XSL-FO, na rzecz „firmowej” składni dostarczanej przez XMLP. W drugiej metodzie wszystkie składnie są dozwolone.

Dostępny jest również oddzielny program *XML Desktop* będący *de facto* instalatorem nakładki na edytor Microsoft Word, umożliwiającym wizualną budowę szablonu raportu w formacie RTF. Nie rozszerza on jednak w żaden sposób funkcjonalności pakietu XMLP, dlatego nie będzie tu omawiany.

1.1. Źródła informacji

Podstawowym źródłem informacji wykorzystywanym w pracy jest dokumentacja firmy Oracle oraz jej strona technologiczna <http://www.oracle.com/technology/index.html>. Niestety, firmowa dokumentacja znana jest z tego, że choć zawiera wszelkie, nawet najdrobniejsze szczegóły opisywanych rozwiązań, to brakuje w niej zwykle zwięzłego przedstawienia istoty omawianych zagadnień. W pracy korzystano głównie z następujących pozycji: [Ora1] [Ora2] [Ora3]. W dalszej części pracy nie są one już nigdzie cytowane, gdyż zawarte w nich informacje przewijają się praktycznie w całym artykule. Pierwsza pozycja jest podstawowym źródłem informacji o wszystkich aspektach działania raportów w środowisku sieciowym. Druga zawiera wiadomości o technicznej stronie konstruowania raportów. Trzecia jest wprowadzeniem do środowiska OC4J, w której działają raporty.

2. Moduł *Reports*

Moduł *Reports*, będący częścią pakietu *Oracle Developer Suite*, był do niedawna jedynym firmowym narzędziem firmy Oracle dedykowanym do tworzenia rozwiązań raportujących, będących (z reguły) częścią większych systemów informatycznych. Starsza wersja pakietu 6i (ciągle bardzo intensywnie wykorzystywana w Polsce i na świecie!) działała w architekturze 2-warstwowej klient-serwer z możliwością pracy w architekturze 3-warstwowej. Ta ostatnia właściwość nie była jednak popularna wśród użytkowników (prawdopodobnie dlatego, że korzystała z hermetycznego środowiska serwera aplikacyjnego) i ponadto była dosyć zawodna od strony technicznej. Ponadto w praktyce budowane raporty mogły korzystać jedynie z danych zgromadzonych w bazie Oracle, które pobierane były jedynie poprzez klasyczne zapytania SQL.

W skład modułu *Reports 6i* wchodziło bardzo wydajne i efektywne środowisko projektowe raportów *Reports Builder*. W najnowszej wersji *Reports 10g* pozostało ono praktycznie niezmienione. W wersji 10g dodano jedynie pewne elementy, które pojawiły się jako nowe możliwości modułu *Reports 10g*. Należą do nich głównie:

- nowe sposoby uruchamiania raportów wynikające ze stosowanej architektury 3-warstwowej w środowisku OC4J,
- obsługa tekstowego formatu zapisu raportu w języku JSP (stary, binarny format RDF, dalej jest obsługiwany),
- obsługa dwóch trybów definicji raportu. Dotychczasowy, zwany obecnie *Paper Layout* (w skrócie PL) i nowy o nazwie *Web Layout* (w skrócie WL). Pliki JSP mogą przechowywać PL, WL lub jednocześnie PL i WL. Pliki RDF jedynie PL,
- możliwość korzystania z nowych, oprócz klasycznych zapytań SQL, źródeł danych.

W dalszej części pracy omówiono dokładniej wymienione wyżej cechy.

2.1. Środowisko pracy *Reports*

Obecnie omówione zostaną możliwe konfiguracje środowiska dewelopera oraz środowiska użytkownika gotowych raportów. W konfiguracji dla dewelopera, po zainstalowaniu modułu *Reports* dysponujemy:

- środowiskiem projektowym *Reports Builder*,
- skonfigurowaną lokalną instancją OC4J z zainstalowanym i skonfigurowanym oprogramowaniem tworzącym właściwe środowisko *Reports* (w warstwie systemowej napisane w technologii serwletów),

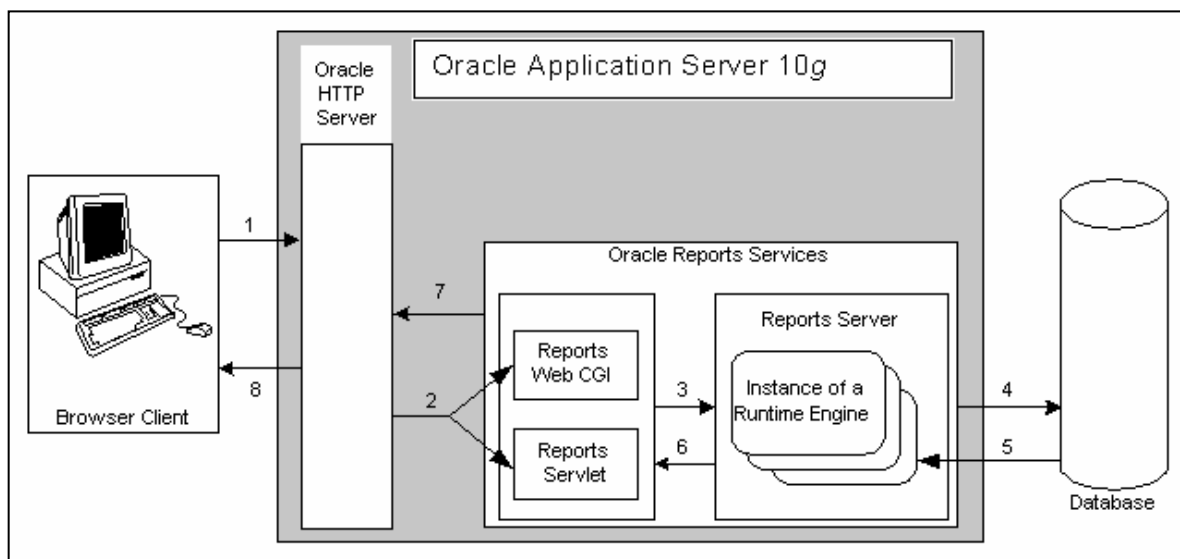
- skonfigurowanym *JDeveloperem* umiejącym skorzystać z lokalnej instancji OC4J w przypadku uruchamiania raportów wprost z poziomu *JDevelopera* oraz z zainstalowaną biblioteką raportowych znaczników JSP (plik *reports_tld.jar*)¹.

W konfiguracji użytkownika (produkcyjnej) nie występują naturalnie narzędzia programistyczne oraz mamy dwie możliwości konfiguracji środowiska udostępniania (ang. *deployment*) i uruchamiania raportów:

- instalacja okrojonej tylko do obsługi raportów (i formularzy) wersji serwera aplikacyjnego pod nazwą *Oracle Application Server 10g Release 2 (10.1.2.0.2) Forms and Reports Services* – w skrócie FRS,
- instalacja pełnego środowiska *Oracle Application Server 10g Release 2 (10.1.2.0.2)*², którego jednym z elementów jest moduł *Reports Services* – w skrócie OAS.

Niezależnie od przyjętego rozwiązania, schemat działania raportów jest zawsze taki sam. Przedstawiono go na Rysunku 1. Główne punkty to:

- (1): wywołanie za pośrednictwem serwera HTTP głównego serwletu modułu *Reports* (o nazwie *rwervlet*),
- (3): parsowanie przez ten serwlet składni wywołania celem konwersji do postaci umożliwiającej obsługę raportu przez motor raportów,
- (4): odwołanie do bazy (lub innego źródła danych) celem pobrania danych opisanych w definicji raportu.



Rys. 1. Schemat uruchamiania raportów w module *Reports*

¹ Interpretacja elementów „projektowych” raportu (zapytania, grupy danych, pola danych, itp.) zapisanych w postaci firmowych znaczników XML (Reports XML Tags) nie jest obsługiwana z poziomu *JDevelopera* (tekst jest skomentowany). Znaczniki te „rozumiane są” jedynie przez środowisko projektowe *Reports Builder*.

² Zwracamy uwagę na numer wersji. W wersji OAS (10.1.2.0.0) NIE było *Reports Services*. W wersji OAS (10.1.3) również ich nie ma. Wiąże się z tym komplikacje; zacytujmy zdanie z serwisu OTN: „If your application uses EJB 3.0 then it will have to be deployed to 10.1.3 - so the solution in your case is to have two instances of OAS running one using 10.1.3 for the Java parts, and one 10.1.2 for the Reports”.

2.1.1. Szybki test działania modułu *Reports*

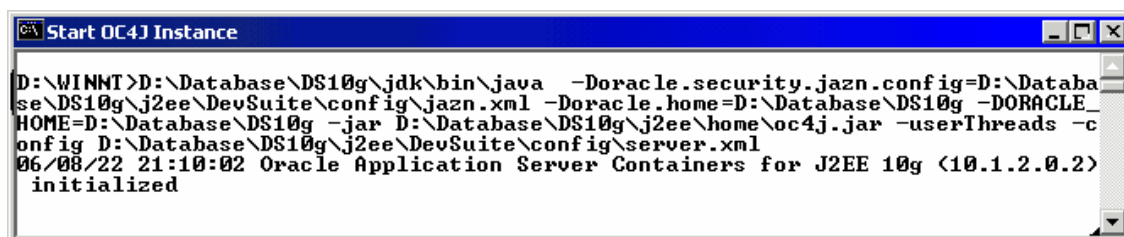
Poniżej pokazano kilka czynności do wykonania, które potwierdzą działanie wymienionych w poprzednim rozdziale elementów środowiska dewelopera i środowiska produkcyjnego.

Uruchamianie lokalnej instancji OC4J

W środowisku dewelopera często będziemy testowali raporty w środowisku internetowym (bezpośrednio z poziomu *Reports Builder*-a też jest to możliwe ale tylko dla jednej grupy raportów tzw. „papierowych”). Wymaga to uruchomienia lokalnej instancji OC4J. W menu Start systemu Windows powinniśmy znaleźć gotowy link do pliku wsadowego. Znajduje się w nim standardowe polecenie uruchamiające kontener OC4J w wersji *standalone* działający w systemie jako proces³.

Uruchamianie instancji OC4J:

```
<DS10g_HOME>\jdk\bin\java
-Doracle.security.jazn.config=<DS10g_HOME>\j2ee\DevSuite\config\jazn.xml
-Doracle.home=<DS10g_HOME>
-DORACLE_HOME=<DS10g_HOME>
-jar <DS10g_HOME>\j2ee\home\oc4j.jar
-userThreads
-config <DS10g_HOME>\j2ee\DevSuite\config\server.xml
```



Rys. 2. Lokalna instancja OC4J uruchomiona jako proces

Zamykanie instancji OC4J:

```
<DS10g_HOME>\jdk\bin\java
-Doracle.security.jazn.config=<DS10g_HOME>\j2ee\DevSuite\config\jazn.xml
-jar <DS10g_HOME>\j2ee\home\admin.jar ormi://localhost:23910 admin manager
-shutdown
```

Ponieważ instancja uruchamiana jest jako proces (a nie jako serwis), więc zamknięcie okna pokazanego na Rysunku 2 będzie skutkowało zamknięciem kontenera OC4J.

Uwaga: dostępna jest niezależna instalacja kontenera OC4J do pobrania ze strony http://download.oracle.com/otn/java/oc4j/1013/oc4j_extended_101300.zip. Teoretycznie można by więc sobie wyobrazić instalację *Reports* na bazie tego kontenera. Dokumentacja nie podaje jednak procedur instalacji w tym środowisku oprogramowania *Reports*.

³ Dostępna wersja OC4J Release 3 posiada już wygodną konsolę administracyjną – dostępną pod adresem: http://<oc4j_host>:port/em. Z najnowszą dostępną wersją Developer Suite, instalowana jest jednak wersja OC4J Release 2, pozbawiona Enterprise Manager-a.

2.1.2. Testowanie działania środowiska dewelopera

Testowanie lokalnej instancji OC4J

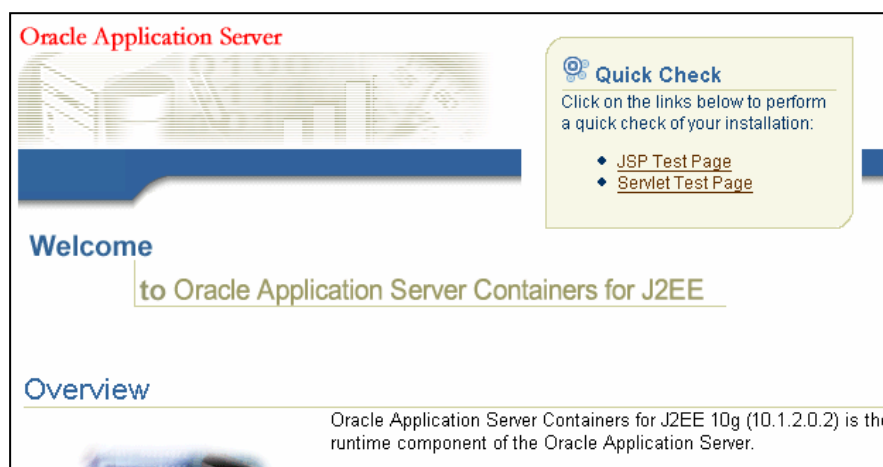
Posługujemy się numerem portu zdefiniowanym w pliku

```
<DS10g_HOME>/install/portlist.ini
```

```
http://localhost:8889/j2ee -- w wersji 10g
```

```
http://localhost:8888 -- w wersji 9i
```

Pojawienie się strony jak na Rysunku 3 świadczy o działającej instancji OC4J z kontenerami obsługującymi strony JSP i serwlety. Znajdziemy na niej również linki do testowych (jeszcze nie związanych w żaden sposób z raportami) stron JSP i testowych serwletów.



Rys. 3. Strona startowa lokalnej instancji OC4J

Testowanie ścieżki względnej do modułu *Reports* w lokalnej instancji OC4J

Sprawdzamy ustawienie względnej ścieżki do usług raportowych na serwerze pod adresem `http://localhost:8889/reports`. Powinna pojawić się statyczna strona „Welcome to Oracle Reports !” umieszczona fizycznie w pliku `<DS10g_HOME>\reports\j2ee\reports_ids\web\index.html`.

Testowanie głównego serwletu *Reports* w lokalnej instancji OC4J

Serwlet `rwservlet` jest głównym „programem”, zdalnie wywoływanym z poziomu przeglądarki, którego zadaniem jest zarówno informowanie o konfiguracji i aktualnym stanie serwera raportów oraz właściwe wykonywanie raportów użytkownika: Po wpisaniu adresu `http://localhost:8889/reports/rwservlet` zobaczymy stronę startową z informacjami o możliwych do wykonania poleceniach serwletu. Na Rysunku 4 pokazano wynik wykonania jednego z dostępnych poleceń `showjobs`: `http://localhost:8889/reports/rwservlet/showjobs?server=rep_tampa`. Widać tam jedno wykonanie (z poziomu *JDeveloper-a*) raportu w postaci pliku `ZAD6.jsp` oraz jedno wykonanie raportu w postaci pliku `test.rdf`.

ORACLE Reports

Stan kolejki serwera Reports

Tryb zabezpieczeń **Bez zabezpieczeń**
Kolejka na serwerze rep_tampa, na Tue Sep 12 13:24:39 CEST 2006
 Aby zlikwidować bieżące (zakolejkowane lub zaplanowane) zadanie, proszę kliknąć na ikonę "zadanie" na następnej stronie. Aby uzyskać wynik z pamięci podręcznej dla uprzednio pokazanego zadania (jeśli dostępne).

Pokaż

Pokaż:

Wynik

| ID zadania | Typ zadania | Nazwa zadania | Stan zadania | Właściciel zadania | Typ wyniku | Nazwa wyniku | Nazwa serwera |
|------------|-------------|---|--------------|--------------------|------------------|-----------------|---------------|
| 126 | report | /Workspace1-Project1-context-root/ZAD6.jsp | ✓ | RWUser | | niezdefiniowane | rep_tampa |
| 125 | report | D:\Database\DS10g\reports\samples\demo\test.rdf | ✓ | RWUser | Pamięć podręczna | niezdefiniowane | rep_tampa |

- ◆ [help](#)
- ◆ [showenv](#)
- ◆ [showjobs](#)
- ◆ [showmap](#)
- ◆ [showmyjobs](#)
- ◆ [showjobid](#)
- ◆ [killjobid](#)
- ◆ [parsequery](#)
- ◆ [showauth](#)
- ◆ [delauth](#)
- ◆ [getjobid](#)
- ◆ [getserverinfo](#)
- ◆ [killengine](#)

Rys. 4. Dostępne polecenia serwletu *rwervlet*. Przykład wykonania polecenia *showjobs* dla serwera *rep_tampa* ze środowiska dewelopera. W tle inne dostępne polecenia (porównaj też przykład z rysunku 6 dotyczący środowiska FRS)

2.1.3. Testowanie działania środowiska FRS

Środowisko FRS (patrz punkt 2.1), może być zarządzane z użyciem *Enterprise Manager / Application Server Control* uruchamianego skryptem `<FRS10g_HOME>\bin\emctl.bat`. Z kolei polecenie `<FRS10g_HOME>\opmn\bin\opmnctl.exe startall` uruchamia właściwe usługi *Reports* działające w środowisku OC4J (patrz serwis *OracleAS10g_FRShome1ProcessManager* na Rysunku 5, *FRS_home1* w nazwie to nazwa *home-a* nadana w czasie instalacji). Obie usługi działają jako serwisy systemowe (środowisko OC4J działa, jak zostało to wcześniej pokazane, jako proces). Na Rysunku 5 pokazano obie usługi w stanie „*Started*”. Do działania raportów jako takich pierwszy oczywiście nie jest wymagany, jednak w praktyce konieczny do wygodnego zarządzania całym środowiskiem. Uwaga: w opisie środowiska FRS poniżej zmieniono standardowy port z 7778 na 80 korzystając ze strony administracyjnej pokazanej na Rysunku 8.

| | | |
|-------------------------------------|---------|--------|
| OracleAS10g_FRS_home1ASControl | Started | Manual |
| OracleAS10g_FRS_home1ProcessManager | Started | Manual |

Rys. 5. Usługa systemowa modułu *Reports* w ramach instalacji typu FRS oraz usługa systemowa zarządzająca całością *Enterprise Manager-a*

Pod adresem `http://localhost/reports/rwervlet` zobaczymy antologiczną stronę do tej pokazanej na Rysunku 4. Wykonując polecenie `http://localhost/reports/rwervlet/getserverinfo` uzyskamy informację o serwerze *Reports* (nazwa serwera to sklejenie przedrostka *rep* z nazwą domenową maszyny, nazwą katalogu, gdzie zainstalowano FRS oraz nazwą *home-a*). Wynik pokazano na Rysunku 6.

ORACLE
Reports

Wydajność Motor

Informacje o serwerze Reports - rep_sz115_AS10g_FRS_home1

Serwer

| | |
|---|---------------------------|
| Nazwa | rep_sz115_as10g_frs_home1 |
| Wersja | 10.1.2.0.2 |
| Tryb zabezpieczeń | Bez zabezpieczeń |
| Tryb wykonywania | Przetwarzane |
| Host | tampa |
| Identyfikator procesu | niezdefiniowane |
| Godzina rozpoczęcia | 2006-09-12 18:54:18 |
| Maksymalny rozmiar kolejki | 1000 |
| Tryb śledzenia | niezdefiniowane |
| Plik śladu | niezdefiniowane |
| Opcje śledzenia | niezdefiniowane |
| Inne serwery Reports w tym samym klastrze | |

Wydajność

Zadania zakończone pomyślnie 45

Rys. 6. Polecenia serwletu *rwservlet*. Przykład wykonania polecenia *getserverinfo* dla serwera *rep_sz115_AS10g_FRS_home1* ze środowiska FRS

Dostępna powinna być również strona umożliwiająca interaktywne uruchamianie raportów JSP oraz RDF: <http://localhost/repdemo>. Bardziej bezpośrednie metody uruchamiania raportów JSP i RDF pokazane zostaną dalej. Na Rysunku 7 pokazano formularz, uruchamiający wybrany raport JSP. W przypadku serwera raportów umieszczonego na innej niż lokalna maszynie, w polu Reports Server, należy go określić. Tu wpis *<In-Process Server>* oznacza lokalny serwer FRS.

Getting Started with Oracle Reports *Test a JSP Web Report*

Welcome to the Oracle Reports JSP-based Web Report Tester.

This HTML form may be used to run a JSP-based Web Report against any Reports Server.

Reports parameters:

Reports Server:

JSP Report:

DB Connect String:

Other Parameters:

Web listener (HTTPD) details:

Web protocol:

Web host:

Web port:

Rys. 7. Interaktywne uruchamianie raportów JSP i RDF w środowisku lokalnego FRS

2.2. Administracja Reports

Środowisko FRS

W praktyce najwygodniej posługiwać się *Enterprise Manager*-em. Po uruchomieniu właściwego serwisu (patrz Rysunek 5) i skierowaniu się na stronę `http://localhost:18100` (konieczna autoryzacja) znajdziemy się na domowej stronie administracyjnej serwera aplikacji w wersji FRS. Jednym z jego elementów jest serwer raportów (tu o nazwie `rep_sz115_AS10g_FRS_home1`). Na Rysunku 8 pokazano ekran startowy z widoczną m.in. aplikacją serwera raportów. Oprócz pełnienia funkcji serwera raportów jest to oczywiście „normalny” serwer OAS, na którym mogą być zainstalowane (ang. *deployment*) dowolne inne aplikacje J2EE.

| Select Name | Status | Start Time | CPU Usage (%) | Memory Usage (MB) |
|---|--------|-------------------------|-------------------|-------------------|
| Forms | ↑ | Aug 20, 2006 1:16:34 AM | 0.00 | 0.00 |
| home | ↑ | Aug 20, 2006 1:16:39 AM | 0.00 | 30.45 |
| HTTP_Server | ↑ | Aug 20, 2006 1:16:39 AM | Not Yet Available | 32.07 |
| OC4J_BI_Forms | ↑ | Aug 20, 2006 1:16:39 AM | 3.32 | 50.88 |
| Reports Server: rep_sz115_AS10g_FRS_home1 | ↑ | N/A | N/A | N/A |
| Web Cache | ↑ | Aug 20, 2006 1:16:39 AM | 0.11 | 40.59 |
| Management | ↑ | Aug 20, 2006 1:25:26 AM | 47.57 | 123.97 |

Rys. 8. Administracja serwera aplikacji. W wersji FRS w jego skład wchodzi serwer raportów zainstalowany w wydzielonej instancji o nazwie *Reports Server: rep_sz115_AS10g_FRS_home1*. Okno logowania do strony administratora sztucznie wklejono dla ilustracji.

Aplikacja *Web Cache*, co widać na Rysunku 8, jest aktywna. Jest ona dostarczana razem z FRS i może być w razie potrzeby uruchamiana / zatrzymywana. Jej opis wykracza jednak poza ramy artykułu. Wchodząc dalej do instancji *Reports Server: rep_sz115_AS10g_FRS_home1* mamy możliwość uruchamiania / zatrzymywania serwera raportów, konfiguracji serwera raportów, edycji jego plików konfiguracyjnych, przeglądanie logów, listy zadań (raportów) do wykonania, itp.

Środowisko OAS

Objętość pracy nie pozwala na zamieszczenie rozważań na temat strukturalnych różnic pomiędzy FRS a usługami *Reports Services* w pełnej instalacji OAS. Różnice leżą głównie w kwestiach konfiguracyjno – wydajnościowych. Od strony użytkowej żadnych różnic nie zaobserwujemy. Wszystkie podane wyżej przykłady będą wyglądały tak samo (w zależności od konfiguracji mogą być jedynie inne numery portów).

Bezpieczeństwo

Mówiąc o bezpieczeństwie mamy na myśli kontrolowany dostęp do modułu *Reports* oraz kontrolowany dostęp do wykorzystywanych (wykonywanych) raportów. Generalnie kontrola odbywać może się w dwóch trybach: z wykorzystaniem *OracleAS Single Sign-On* (SSO) oraz bez SSO. Kwestie pracy w środowisku SSO, obejmujące zagadnienia bezpieczeństwa modułu *Reports* oraz bezpieczeństwa danych, z których *Reports* korzystają, wykraczają poza ograniczony zakres pracy. W trybie bez SSO można uwierzytelnić⁴ użytkownika przy każdym wywołaniu raportu (taki też sposób użyto w dalszej części pracy, pokazując przykłady wywołań raportów w przeglądarce). Jest to jednak najprostszy sposób i w praktyce dosyć uciążliwy. Skorzystać można jednak z obsługiwanego przez *Reports* kodowanego cookie o nazwie *USERID*, który na czas sesji z raportami przechowywać będzie potrzebne dane na lokalnym dysku użytkownika. Innym dostępnym rozwiązaniem jest skorzystanie z pliku *cgicmd.dat*⁵ (tzw. *mapping file*), przechowywanego na serwerze. Gdy plik ten jest stosowany, jego nazwa jest jednym z parametrów wywołania raportu.

2.3. Budowa raportów

Obecnie pokazany zostanie w skrócie proces konstruowania raportów w środowisku deweloperskim. Budując raporty do dyspozycji mamy wspomniane już wcześniej środowisko projektowe *Reports Builerd* oraz (od wersji 9i) środowisko *JDeveloper*. Pierwsze jest podstawowym narzędziem dewelopera. Drugie, mimo niewątpliwiej promocji prowadzonej przez firmę Oracle, wykorzystywane jest niestety w niewielkim stopniu jeśli chodzi o budowę raportów.

W środowisku projektowym *Reports Builder* dostępny jest graficzny kreator raportów. Jako stosunkowo prosty w obsłudze, nie będzie on tu omawiany. Znajomość natomiast zasad ręcznej budowy raportów (czyli bez użycia kreatorów) jest niezbędna do: 1. budowy niestandardowych raportów, które trudno uzyskać pracując jedynie z kreatorem, 2. ręcznej edycji raportów wygenerowanych automatycznie celem dostosowania ich do szczegółowych wymagań użytkownika.

Poniżej omówiono i zilustrowano podstawowy ciąg czynności prowadzący do skonstruowania raportu. Omawiany w dalszej części pracy pakiet XMLP, charakteryzuje się dużym podobieństwem do *Reports Builerd-a*. Inne są jedynie używane tam technologie.

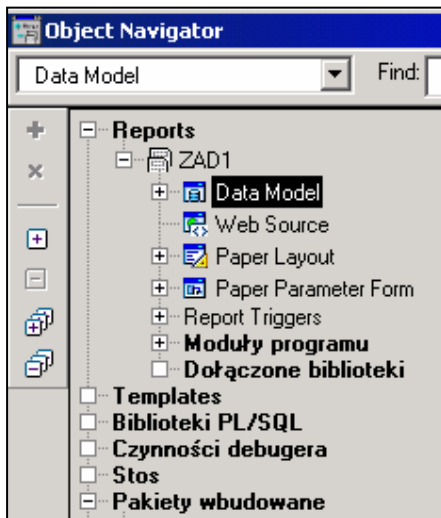
Model danych

Pierwszym krokiem jest wybór danych, na których oprzemy nasz raport (Rysunek 9). Na początek założymy, że dane pobieramy zapytaniem SQL. Niezależnie jednak z jakiego źródła korzystamy, istotną i bardzo użyteczną z praktycznego punktu widzenia cechą *Reports Builerd-a* jest możliwość dowolnego grupowania danych w sposób graficzny (przez przeciąganie myszką wybranych danych). Zapytanie *Q_1* jest zwykłym zapytaniem *SELECT* bez grupowania *GROUP BY*. Podział pracowników na wydziały wykonano więc pośrednio, wyprowadzając dane o pracownikach z poza grupy *G_Wydzial*.

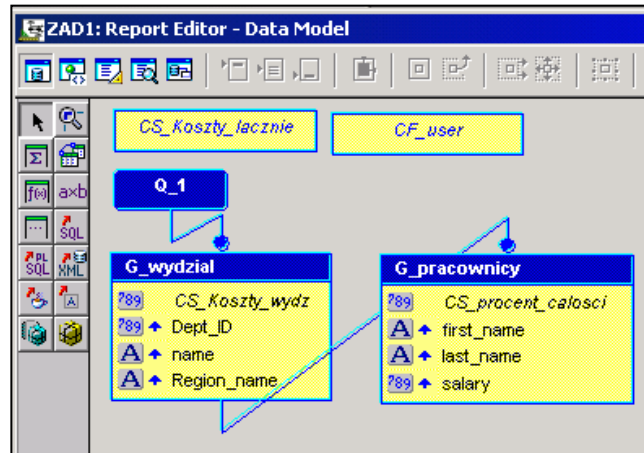
Dużą zaletą jest również możliwość korzystania w jednym raporcie z dowolnie dużej liczby zapytań, których wyniki można ze sobą łączyć na potrzeby na przykład łatwej budowy raportów w tzw. układzie macierzowym lub korzystania z wyników jednego zapytania w drugim. Przykład pokazano na Rysunku 11

⁴ *authentication* (uwierzytelnienie użytkownika). Nie mylić z *authorization* (autoryzacja, uprawnienie do wykonania raportu w środowisku *Reports*). To ostatnie zagadnienie pomijamy w pracy.

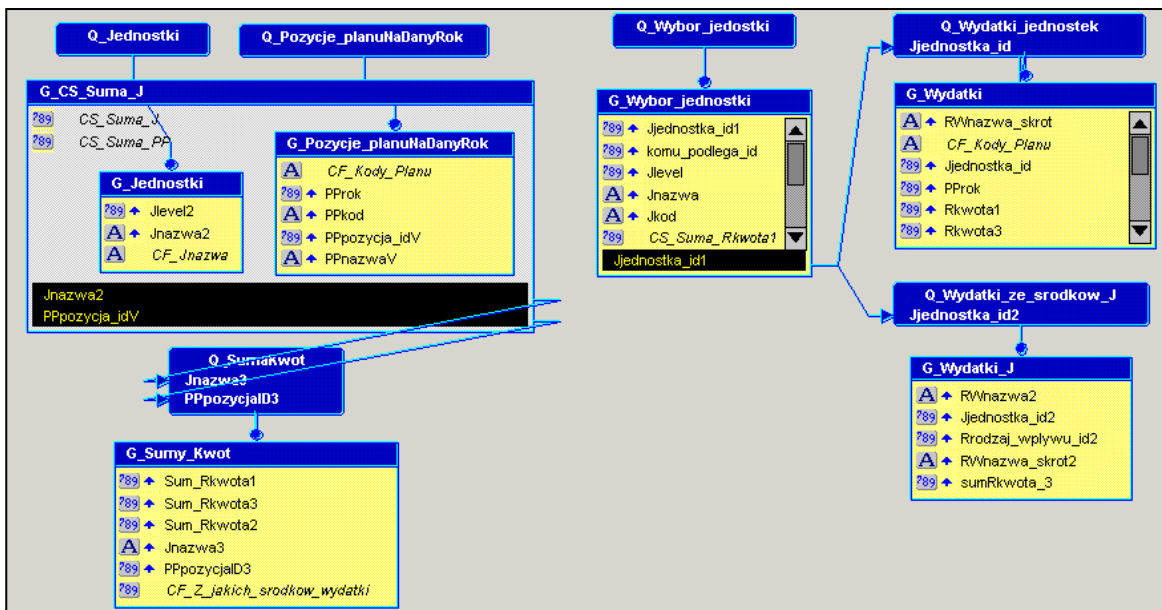
⁵ Plik ten służy też innym, nie tylko związanym z bezpieczeństwem, celom.



Rys. 9. Reports Builder. Pierwsze kroki budowy raportu. Model danych



Rys. 10. Reports Builder. Wizualizacja pobranych zapytaniem danych po ich pogrupowaniu



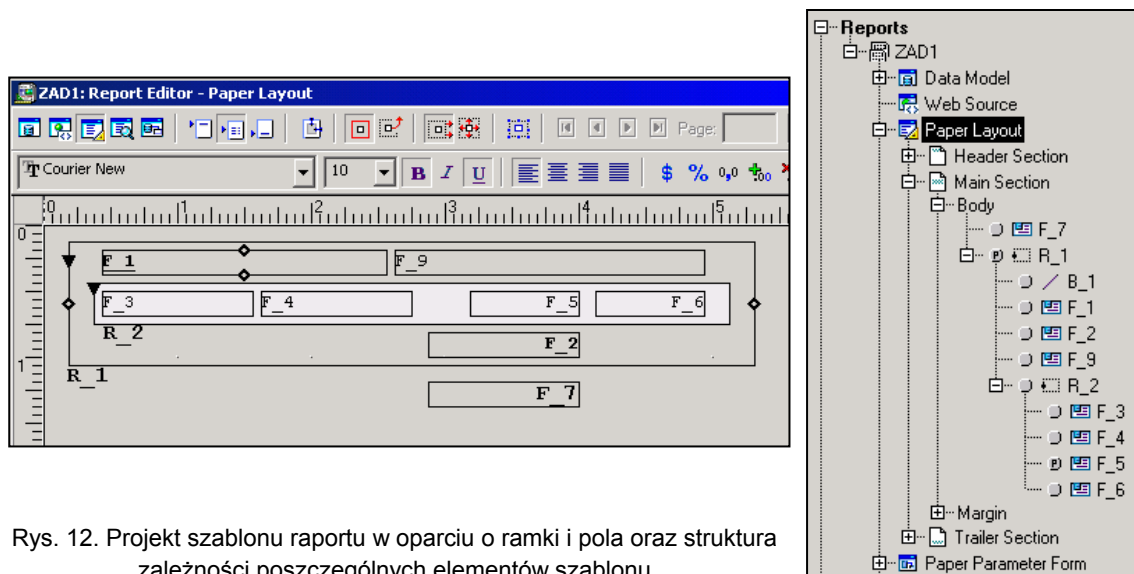
Rys. 11. Łączenie wyników prostych zapytań w celu uzyskania bardziej złożonych raportów

Raporty mogą korzystać nie tylko z danych pobieranych za pomocą zapytania SQL. Możliwe jest również korzystanie na przykład z „płaskich” plików tekstowych (konieczne zdefiniowanie formatu danych w pliku `<FRS10g_HOME>\reports\conf\textpds.conf`). Dane pobierać można z arkuszy kalkulacyjnych np. w formacie Excel, korzystając w tym wypadku z tzw. mostu (ang. *jdbc-odbc bridge*) (patrz analogiczny jak wyżej plik `<FRS10g_HOME>\reports\conf\jdbcpds.conf`). Korzystanie z danych XML również nie stanowi problemu. Obsługiwane są również usługi sieciowe (ang. *Web Services*). Nie są one oczywiście źródłami danych, choć można je traktować jako źródła informacji. Korzystanie z nich wymaga napisania (korzystając np. z *JDevelopera*) odpowiedniej klasy Java (ang. *Web Service Stub*) a następnie wywołanie jej z poziomu odpowiedniego wrapper-a PL/SQL, który zostanie utworzony automatycznie przez moduł *Reports*.

Szablon raportu „papierowego”

Kolejnym krokiem jest definicja szablonu (układu) raportu. Na Rysunku 12 pokazano szablon raportu „papierowego” (ang. *Paper Layout*) oraz dla ilustracji strukturę zależności poszczególnych jego elementów. Szablon raportu to nic innego jak graficzny opis sposobu rozmieszczenia w raporcie danych pobranych na etapie definiowania Modelu Danych. Główne wykorzystywane tu elementy to tzw. ramki (ang. *Frames*) oraz pola (ang. *Fields*). Ramki (R_1 , R_2) służą określeniu przestrzennego umiejscowienia pochodzących z pól (np. F_1 , F_4) danych. Kojarzone są one z grupami danych (na przykład ramkę R_1 skojarzono z grupą $G_{Wydzial}$). Ramki mogą zawierać się w sobie (R_2 zawiera się w R_1).

Dokładnie wg. tych samych zasad zaprojektowano sposób budowy szablonów raportów w pakiecie *XMLP*. Role ramek i pól pełnią tam odpowiednie znaczniki XSL wpisane w pliku RTF.



Rys. 12. Projekt szablonu raportu w oparciu o ramki i pola oraz struktura zależności poszczególnych elementów szablonu

Możliwości formatowania elementów szablonu raportu „papierowego”

Formatowanie „statyczne” (np. kolory elementów, grubości ramek) odbywa się bezproblemowo. Formatowanie „dynamiczne” poszczególnych elementów raportu w czasie jego wykonywania odbywa się natomiast poprzez skojarzenie z formatowanym elementem krótkiego kodu (trygera) w języku PL/SQL. Przykładowo z polem F_5 skojarzono funkcję $F_5FormatTrigger$, której przeznaczenia łatwo się domyślić.

```
function F_5FormatTrigger return boolean is
begin
  if (:salary < '1000') then
    srw.set_font_weight(SRW.BOLD_WEIGHT);
    srw.set_text_color('red');
  end if;
  return (TRUE);
end;
```

Opisane możliwości formatowania to często krytykowana cecha raportów, gdyż nawigacja pomiędzy bardzo rozproszonym w strukturze szablonu kodem PL/SQL jest w praktyce dosyć trudna do ogarnięcia. Niestety XMLP dziedziczy te niedogodności. Kod formatujący (tu: znaczniki XSL) są również porzucane w całym pliku RTF i jeszcze trudniej dostępne, gdyż w większości są one dodatkowo poukrywane w polach formularzy tekstowych. Nie istnieje natomiast żaden sposób ich

wizualizacji w strukturze szablonu, jak ma to miejsce w środowisku *Reports Builder*, co pokazano po prawej stronie Rysunku 12.

Szablon raportu „internetowego”

Graficzny sposób konstruowanie szablonu, który możliwy jest w przypadku raportu „papierowego” (ustawianie pól, ramek, itp.) nie jest możliwy w tzw. raportach internetowych (ang. *Web Layout*). W tej sytuacji tworzenie raportów w zasadzie wymusza opieranie się na kreatorach. Po ich wstępnym utworzeniu (kreatorem), dalsza ręczna ich edycja jest możliwa jedynie w stosunkowo niewielkim zakresie⁶ i w praktyce sprowadza się do ... ponownego korzystania z kreatora. Raporty internetowe zapisywane są z reguły w pliku JSP, który może być dowolnie edytowany. Wprowadzanie więc do nich zmian w praktyce sprowadza się do ręcznej edycji wygenerowanego pliku.

Edycja raportu JSP wymaga jednak: 1. znajomości podstawowych technik internetowych, 2. biegłości w programowaniu w językach Java / JSP / HTML, 3. poznania biblioteki *Reports JSP Custom Tags*, 4. poznania biblioteki *Reports XML Tags*. Nie jest to więc trywialne zadanie, choć dość często jest to tak właśnie przedstawiane na stronach producenta.

Poniżej pokazano szkielet raportu JSP. `prefix="rw"` oznacza prefix znaczników *Reports JSP Custom Tags*. Znacznik `<rw:objects ...>` rozpoczyna definicję raportu (np. zdefiniowane w raporcie zapytania, grupy, itp.) plus ew. opis *Paper Layout* gdy występuje. Począwszy od znacznika `</rw:dataArea ...>` zaczyna się pobieranie danych z bazy i ich rozmieszczanie na raporcie.

```
<%@ taglib uri="/WEB-INF/lib/reports_tld.jar" prefix="rw" %>
<!--
<rw:report id="report" parameters="server=rep_tampa&userid=summit2/summit2">
  <rw:objects id="objects">
  </rw:objects>
-->
<html><head>
<meta name="GENERATOR" content="Oracle 9i Reports Developer"/>
<title> Your Title </title>

<rw:style id="yourStyle">
  <!-- Report Wizard inserts style link clause here -->
</rw:style>

</head><body>

  <rw:dataArea id="yourDataArea">
  </rw:dataArea>

</body></html>
<!--
</rw:report>
-->
```

Przykładowy fragment zawartości znacznika `<rw:objects ...>` znajduje się poniżej. Wewnątrz niego użyto wielu znaczników *Reports XML Tags*, „rozumianych” jedynie przez *Reports Builder*-a. Na Rysunku 13 pokazano graficzną ilustrację opisywanego w pliku JSP modelu danych.

```
<%@ taglib uri="/WEB-INF/lib/reports_tld.jar" prefix="rw" %>
```

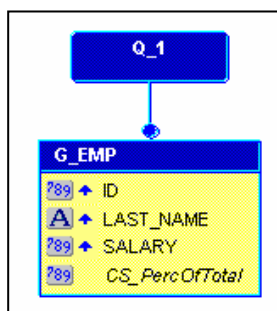
⁶ Raport może zawierać w sobie zarówno *Paper Layout* jak i *Web Layout*. Oba szablony są niezależne od siebie, więc edycja jednego nie zmienia drugiego.

```

<rw:report id="report" parameters="server=rep_tampa&userid=summit2/summit2">
<rw:objects id="myObjects">
<report DTDVersion="9000010" name="myReport">
  <data>
    <dataSource name="Q_1">
      <select>
        SELECT E.ID, E.LAST_NAME, E.SALARY FROM S_emp E WHERE E.ID > 10
      </select>

      <group name="G_EMP">
        <dataItem name="ID"/>
        <dataItem name="LAST_NAME"/>
        <dataItem name="SALARY"/>
        <summary name="CS_PercOfTotal" source="SALARY"
          function="percentOfTotal" reset="G_EMP" compute="report">
          </summary>
        </group>
      </dataSource>
    </data>
  </report>
</rw:objects>

```



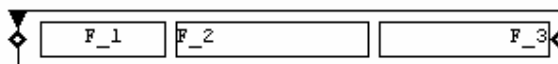
Rys. 13. Model danych raportu w postaci graficznej. Model ten zapisano w pliku JSP powyżej

Przykładowy fragment zawartości znacznika `<rw:dataArea ...>` zamieszczono poniżej. Obok kodu pokazano, jedynie dla ilustracji, odpowiednik „w notacji” *Paper Layout*.

```

<rw:dataArea id="yourDataArea">
<table border="1" width="100%">

```



```

  <!-- Header -->

```

```

<tr>
  <th <rw:id id="EmpId"/>> Employee # </th>
  <th <rw:id id="EmpName"/>> Employee Name </th>
  <th <rw:id id="Total"/>> % of Total: </th>
</tr>

```

```

  <!-- Body -->

```

```

  <rw:seq name="bgcolor" seq="#bbbbbb, #ffffff"/> def. sekwencji co drugi
  wiersz w innym kolorze

```

```

  <rw:foreach id="G_EMP_ID" src="G_EMP"> pobieramy wiersze z grupy G_EMP

```

```

  <tr bgcolor="<rw:seqval ref="bgcolor" op="nextval"/>"> dane z sekwencji

```

```

    <td headers="<%= EmpId %>">

```

```

    <rw:field id="F_1" src="ID"> F_1 </rw:field>

```

```

  </td>

```

kolumna ID (z grupy G_EMP), pole

F_1

```

<td headers="<%= EmpName %>">
<rw:field id="F_2" src="LAST_NAME"> F_2 </rw:field>
</td>

<td headers="<%= Total %>">
<rw:field id="F_3 src="CS_PercOfTotal"> F_3 </rw:field>
</td>

</tr>
</rw:foreach>
...
</rw:dataArea>

```

2.4. Udostępnianie i uruchamianie Raportów

Załóżmy, że mamy do dyspozycji trzy raporty. Pierwszy, `R1.rdf`, zawiera tylko *Paper Layout*. Drugi raport `R2.jsp` zawiera tylko *Web Layout*. Trzeci raport `R3.jsp` zawiera zarówno *Paper Layout* jak i *Web Layout*. Są to wszystkie możliwe typy raportów.

Raporty JSP zawierające *Paper Layout* wykonywane są za pośrednictwem serwletu *rwservlet*. Raporty JSP z *Web Layout* wykonywane są za pośrednictwem kontenera JSP lub za pośrednictwem serwletu *rwservlet*. W przypadku raportów JSP, druga metoda daje większe możliwości wpływania na format raportu, sposób jego dystrybuowania, itp.

Każdy raport to po prostu plik JSP lub RDF. Udostępnienie raportów (ang. *deployment*) polega więc na skopiowaniu ich we właściwe miejsca na serwerze aplikacyjnym. Przykładowo dla instalacji typu FRS (pod Windows) miejscami takimi są:

- dla plików JSP z *Web Layout* dostępnych pod ścieżką względną `/reports/filename.jsp`:
`<FRS10g_HOME>\j2ee\OC4J_BI_Forms\applications\reports\web\examples\Tools`
- dla plików RDF i JSP dostępnych pod ścieżką względną `/reports/rwservlet?report=filename.jsp`:
`<FRS10g_HOME>\reports\samples\demo\`

Podane powyżej lokalizacje są lokalizacjami poinstalacyjnymi. Można je zmieniać w plikach konfiguracyjnych.

W pracy, z uwagi na jej ograniczoną objętość, nie rozważamy kwestii ogólnych aplikacji J2EE, których fragmentem mogą być na przykład raporty JSP. W takim przypadku należałoby opisać dokładnie kwestie związane z tworzeniem aplikacji J2EE w środowiska OC4J. W największym skrócie aplikacja J2EE tworzona jest jako plik WAR (*Web Application Archive*). Następnie jest ona deployowana jako plik EAR (*Enterprise Archive*). Proces deployowania może z kolei odbywać się ręcznie (dość trudne zadanie) lub z poziomu aplikacji internetowej *Enterprise Manager*.

W przypadku, gdy raport zawiera odwołania do np. zewnętrznych klas Java, należy zadbać o widoczność w raporcie tych klas. Przykładowo dla raportu JSP z *Paper Layout* należy ustawić zmienną `CLASSPATH` w pliku `ORACLE_HOME\reports\conf\server_name.conf`. Dla raportu JSP z *Web Layout* należy dodać klasę lub plik JAR do pliku WAR lub ustawić właściwe `CLASSPATH` kontenera J2EE (bo w nim wykonuje się raport JSP *Web Layout*).

Przykłady wywołań

W jednym z wcześniejszych rozdziałów pokazano sposób uruchamiania raportu z poziomu gotowego formularza (patrz Rysunek 7), w którym podaje się wszystkie niezbędne dane potrzebne

do wykonania danego raportu. Obecnie pokażemy jak wykonywać raporty podając stosowne polecenie ręcznie. Wszędzie posługujemy się domyślnie ustawionymi ścieżkami względnymi. Portem raportów jest standardowy port 80 (zmieniono z domyślnej wartości, jaką jest 7778). Używany jest również wszędzie najprostsz (bezpośredni) sposób uwierzytelniania użytkownika. Pomijamy zagadnienie użycia tzw. pliku mapującego (ang. *key map file*) <FRS_HOME>\reports\conf\cgicmd.dat, który umożliwia całkowite ukrycie parametrów w wywołaniach raportów. W przypadku nie podania (lub podania niepełnych) danych uwierzytelniających, wywołana zostanie standardowa strona z formularzem do logowania. Raporty mogą być wykonywane w następujący sposób:

- jako klasyczne w sensie standardu J2EE wywołania plików JSP. Na ekranie pojawi się więc zawartość *Web Layout* wywoływanego raportu, np.

http://localhost/reports/examples/Tools/R3.jsp?userid=summit2/summit2@ORACLE10_SID

- za pośrednictwem głównego serwletu *rwservlet* z plikiem JSP jak parametrem. Plik ten zawiera jedynie *Paper Layout*. Na ekranie przeglądarki pojawi się więc zawartość *Paper Layout*. *Web Layout* (nawet gdyby był obecny w pliku) nie będzie widoczny. Fragmenty raportów pokazano na Rysunku 14. Przykładowo:

http://localhost/reports/rwservlet?report=R2.jsp&desformat=html&destype=cache&userid=summit2@ORACLE10_SID

http://localhost/reports/rwservlet?report=R2.jsp&desformat=pdf&destype=cache&userid=summit2@ORACLE10_SID

Inne możliwości (wydruk do pliku PDF):

http://localhost/reports/rwservlet?report=R2.jsp&desformat=pdf&destype=file&userid=summit2@ORACLE10_SID&desname=D:\stores.pdf

Inne możliwości (przekazywanie parametrów; gdy są wymagane przez raport)

http://localhost/reports/rwservlet?report=R2.jsp&desformat=pdf&destype=cache&userid=summit2@ORACLE10_SID&p_1=wartosc&p_2=wartosc

Inne możliwości (wysłanie raportu pocztą):

http://localhost/reports/rwservlet?report=R2.jsp&desformat=pdf&destype=mail&userid=summit2@ORACLE10_SID&desname=J.Gramacki@wp.pl

Aby ostatni przykład wykonał się poprawnie, w pliku konfiguracyjnym modułu *Reports* należy wpisać adres serwera SNMP.

Plik konfiguracyjny: <FRS10g_HOME>/reports/conf/rep_sz115_as10g_frs_home1.conf
Wpis: <!--pluginParam name="mailServer">%MAILSERVER_NAME%</pluginParam-->

Sales (Region: North America)

| | | | |
|----------|--------|-----------------|--|
| Magee | Colin | 1400 | |
| Nagayama | Midori | 1400 | |
| | | 2 800,00 | |

Operations (Region: North America)

| | | | |
|----------|---------|-----------------|--|
| Smith | George | 940 | |
| Maduro | Elena | 1400 | |
| Jrguhart | Molly | 1200 | |
| Ngao | LaDoris | 1450 | |
| | | 4 990,00 | |

Administration (Region: North America)

| Id | Last Name | Last N |
|----|-----------|-----------|
| 11 | Magee | Magee |
| 12 | Giljum | Giljum |
| 13 | Sedeghi | Sedeghi |
| 14 | Nguyen | Nguyen |
| 15 | Dumas | Dumas |
| 16 | Maduro | Maduro |
| 17 | Smith | Smith |
| 18 | Nozaki | Nozaki |
| 19 | Patel | Patel |
| 20 | Newman | Newman |
| 21 | Markarian | Markarian |
| 22 | Chang | Chang |
| 23 | Patel | Patel |
| 24 | Dancs | Dancs |
| 25 | Schwartz | Schwartz |

Rys. 14. Fragment raportu *R1.rdf* zawierającego tylko *Paper Layout* wykonany z poziomu przeglądarki WWW (po lewej), oraz fragment raportu *R2.jsp* zawierającego tylko *Web Layout* (po prawej)

- za pośrednictwem głównego serwletu *rwservlet* z plikiem RDF jak parametrem.

Wywołania będą analogicznie jak powyżej. Jedynie parametrem zamiast pliku JSP będzie plik RDF. Pliki RDF mogą zawierać jedynie szablony *Paper Layout*. Przykładowo:

```
http://localhost/reports/rwservlet?report=R1.rdf&desformat=html&destype=cache&userid=submit2@ORACLE10_SID
```

- inne metody, które zostaną tu jedynie zasygnalizowane:

1. poprzez lokalnie zainstalowany *Reports Client* o nazwie *rwclient* parsujący linię komend i przesyłający ją do serwera raportów. Jest to jednak metoda wychodząca już z użycia, gdyż wymaga instalacji oprogramowania na stacji klienta.
2. lokalnie zainstalowany *Reports Runtime* o nazwie *rwruntime* korzystający z serwera w wersji *in-process server* (użyteczny w zasadzie tylko do lokalnego testowania raportów).
3. translator CGI *rwcgi.exe* będący pośrednikiem pomiędzy serwerem webowym a serwerem raportów.

3. Pakiet XMLP

XML Publisher jest nową propozycją firmy Oracle służącą w założeniach do maksymalnie prostej i efektywnej budowy systemów raportujących. Celem artykułu nie jest dokładne przedstawienie tego środowiska. Jego szczegółowy opis zamieszczono w pracach [Tra06][Rei06]. Tutaj zostaną przede wszystkim sformułowane pewne uwagi natury porównawczej pakietu XMLP z modułem *Reports*.

Pakiet XMLP, mimo elementów zasadniczo różniących go od modułu *Reports*, w istocie opiera się na prawie takich samych założeniach „konstrukcyjnych”. Inne są środki implementujące te założenia, istota pozostaje jednak niezmienną. Na Rysunku 15 pokazano najważniejszy dla dewelopera element środowiska projektowania raportów – fragment pliku RTF. Widać tam elementy analogiczne do ramek i pól występujących w środowisku projektowym *Reports Builder*. Elementy te to pola tekstowe edytora (przykładowe treści zawarte wewnątrz nich pokazano w ramach obja-

śnien). Na Rysunku 16 pokazano odpowiadający tabeli z Rysunku 15 kod w postaci „źródłowej” (w takiej postaci, z uwagi na ograniczone możliwości formatowania, w praktyce rzadko stosowane). Na Rysunku 17 pokazano wynik wykonania raportu w środowisku XMLP. Merytorycznie jest to ten sam raport, co przedstawiony na Rysunku 14. Na Rysunku 18 pokazano fragment źródła danych dla raportu, którym w tym przypadku jest plik XML (umieszczony na serwerze XMLP/OC4J pod adresem *http://localhost:15101/R1.xml*).

| | | | |
|---|--------------|--|-----|
| <code><?for-each:G_WYDZIAL?></code> | | <code><?if:SALARY<1000?></code> | |
| FE SORT DNAME_RNAME | | | |
| FE IF1 LAST_NAME | FIRST_NAME | IF SALARY EIF | EIF |
| IF2 LAST_NAME | FIRST_NAME | IF SALARY EIF | EIF |
| | | IF SALARY EIF | EIF |
| | | IF SALARY EIF | EFE |
| | | SUM1 | EFE |
| | Suma: | SUM2 | |
| | | <code><?end for-each?></code> | |

Rys. 15. Tabela w pliku RTF z umieszczonymi polami tekstowym, wewnątrz których znajdują się kody w języku XSL (wymagana zmodyfikowana przez XMLP składnia)

```

<?for-each: G_WYDZIAL?>
  <?sort:NAME?>
  <?NAME?> <?REGION_NAME?>
  <?for-each: G_PRAOWNICZY?>
    <?LAST_NAME?> <?FIRST_NAME?>
    <?if:SALARY<1000?><?SALARY?><?end if?>
    <?if:SALARY>=1000?><?SALARY?><?end if?>
  <?end for-each?>
  <?sum(SALARY)?>
<?end for-each?>
<?sum(SALARY)?>

```

Rys. 16. Kody w języku XSL odpowiadające merytorycznie zawartości pól tekstowych w pliku RTF z Rysunku 15.

| | | |
|--|--------------|--------------|
| Administration(Region: North America) | | |
| Ropeburn | Audry | 1550 |
| Velasquez | Carmen | 2500 |
| | | 4050 |
| Operations(Region: North America) | | |
| Smith | George | 940 |
| Maduro | Elena | 1400 |
| Urguhart | Molly | 1200 |
| Ngao | LaDoris | 1450 |
| | | 4990 |
| Sales(Region: North America) | | |
| Magee | Colin | 1400 |
| Nagayama | Midori | 1400 |
| | | 2800 |
| | Suma: | 11840 |

Rys. 17. Wynik wykonania raportu w środowisku XMLP. Definicja raportu w pliku RTF, którego zawartość pokazano na Rysunku 15

```

<R1>
  <G_WYDZIAL>
    <DEPT_ID>31</DEPT_ID>
    <NAME>Sales</NAME>
    <REGION_NAME>(Region: North America)</REGION_NAME>
    <G_PRACOWNICY>
      <FIRST_NAME>Colin</FIRST_NAME>
      <LAST_NAME>Magee</LAST_NAME>
      <SALARY>1400</SALARY>
      <CS_PROCENT_CALOSCI>50</CS_PROCENT_CALOSCI>
    </G_PRACOWNICY>
    <G_PRACOWNICY>
      <FIRST_NAME>Midori</FIRST_NAME>
      <LAST_NAME>Nagayama</LAST_NAME>
      <SALARY>1400</SALARY>
      <CS_PROCENT_CALOSCI>50</CS_PROCENT_CALOSCI>
    </G_PRACOWNICY>
    <CS_KOSZTY_WYDZ>2800</CS_KOSZTY_WYDZ>
  </G_WYDZIAL>
  <G_WYDZIAL>
  ...

```

Rys. 18. Źródło danych dla raportu z Rysunku 17

Załóżmy, że wynik działania raportu z Rysunku 17 chcielibyśmy przedstawić na wykresie w postaci podsumowania kosztów płacowych wydziałów. Tego typu zadania wymagają zdecydowanie najwięcej pracy w pakiecie XMLP. Na Rysunku 19 pokazano konieczny do wpisania w pliku RTF kod źródłowy, który wygeneruje na stronie WWW raportu prosty wykres słupkowy o domyślnym wyglądzie. Modyfikacja tego wyglądu to oczywiście konieczność wpisania dodatkowego kodu.

```

chart:
<Graph graphType = "BAR_HORIZ_CLUST">
<Title text="Koszty płacowe wydziałów" visible="true"
  horizontalAlignment="CENTER"/>
<Y1Title text="Suma zarobków" visible="true"/>
<O1Title text="Wydział" visible="true"/>

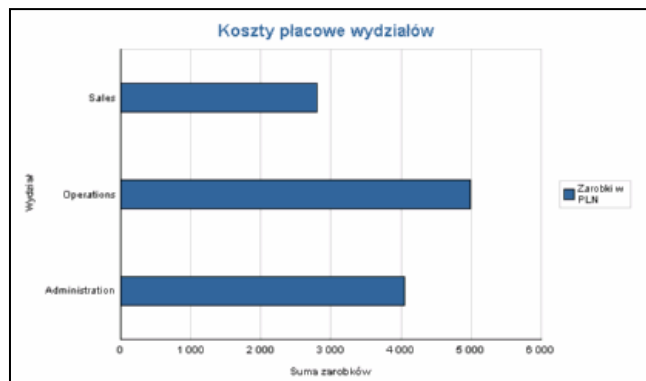
<LocalGridData colCount="{count(//G_WYDZIAL)}" rowCount="1">

<RowLabels>
  <Label>Zarobki w PLN</Label>
  <Label>Procent</Label>
</RowLabels>

<ColLabels>
  <xsl:for-each select="//G_WYDZIAL">
  <Label>
  <xsl:value-of select="NAME"/>
  </Label>
  </xsl:for-each>
</ColLabels>

<DataValues>
<RowData>

```



```
<xsl:for-each select="//G_WYDZIAL">
  <Cell>
    <!--<xsl:value-of select=sum("SALARY") />-->
    <xsl:value-of select="CS_KOSZTY_WYDZ"/>
  </Cell>
</xsl:for-each>
</RowData>
</DataValues>

</LocalGridData>
</Graph>
```

Rys. 19. Kod źródłowy w pliku RTF odpowiedzialny za wykonanie wykresu słupkowego danych z raportu na Rysunku 17. Wykres wykonano z ustawieniami domyślnymi (np. bardzo małe i nieczytelne etykiety, jednolity kolor słupków, itp.)

3.1. Podobieństwa i różnice

Wymieńmy teraz główne podobieństwa i różnice modułu *Reports* i pakietu XMLP.

Główne cechy wspólne to:

- architektura 3-warstwowa z motorem raportowym zaimplementowanych w technologii serwetów w środowisku OC4J,
- ten sam ciąg czynności zmierzający do utworzenia raportu rozpoczynający się definicją (pobranie) danych, dalej następuje definicja ewentualnych parametrów raportu, aż po definicję szablonu (układu) raportu,
- graficzny szablon raportu XMLP oparty jest na bardzo podobnych pomysłach jakie obowiązywały w module *Reports*. Po bliższym zapoznaniu się z pakietem XMLP, okazuje się, że podstawowe elementy jakimi w module *Reports* były ramki (ang. *frames*) i pola (ang. *fields*) wstawiane / edytowane z poziomu *Reports Builder-a*, w XMLP zastąpione zostały przez odpowiednie znaczniki XSL⁷ wstawiane / edytowane w pliku tekstowym w formacie RTF,

Główne różnice to:

- zamiast instalowanego na każdej stacji dewelopera zintegrowanego pakietu IDE *Reports Builder*, posługujemy się aplikacją obsługiwaną z poziomu przeglądarki WWW (bardzo prosta w użyciu) oraz dowolnym edytorem obsługującym format RTF,
- dynamiczna modyfikacja wyglądu elementów raportu w czasie jego wykonywania odbywa się w module *Reports* za pomocą kodów PL/SQL. W XMLP za pomocą znaczników XLS (XLS-FO). W obu rozwiązaniach są to z reguły bardzo krótkie wstawki, które występują w dużych ilościach,
- w module *Reports* obsługiwane są dwa formaty raportów: tradycyjny RDF (format binarny) oraz tekstowy JSP. Plik w formacie RDF jest jednocześnie interpretowany przez serwet *rw-servlet* w czasie uruchamiania raportu. Plik w formacie JSP zawiera wszystkie elementy konieczne do „wykonania go” w odpowiednik kontenerze instancji OC4J. W pakiecie XMLP obsługiwanym formatem zapisu raportu jest plik XML (bardzo prosty). Jest on interpretowany przez serwet *xmlpserver* zainstalowany w lokalnej instancji OC4J lub w zewnętrznym kontenerze Tomcat. Aby korzystać z zewnętrznego kontenera konieczna jest ręczna instalacja w nim oprogramowania tworzącego środowisko XMLP,

⁷ W wielu przypadkach wymaga się używania zmodyfikowanej przez XMLP składni tych znaczników. Oryginalne znaczniki XSL można używać, ale nie w każdej sytuacji.

- instalowana wraz z modułem *Reports* lokalna (*standalone*) instancja OC4J nie zawiera pakietu *Enterprise Manager / Application Server Control*. Analogiczna lokalna instancja instalowana wraz z XMLP zawiera go. W praktyce korzystanie z *Enterprise Manager*-a rzadko będzie jednak tak na prawdę potrzebne,
- korzystając z modułu *Reports* mamy możliwość instalacji produkcyjnej wersji OAS okrojonej tylko do obsługi aplikacji *Reports / Forms* (wersja FRS) lub instalacji usług raportowych w pełnej instalacji OAS-a. W obecnej wersji XMLP Oracle nie udostępnia takiej możliwości ani nie podaje sposobu np. doinstalowania XMLP do istniejącej już instalacji OAS-a.

3.2. Porównanie narzędzi deweloperskich

- w module *Reports* łatwiej ogarnąć krótkie wstawki PL/SQL, których w rzeczywistym raporcie może być bardzo dużo (dostępne jest rozwijalne drzewo wszystkich elementów występujących w danym raporcie – Rysunek 12). Pracując z plikami RTF takiej wizualizacji nie ma. W XMLP, szczególnie gdy korzystamy z pól tekstowych formularzy, paca z dużą ich ilością może być uciążliwa,
- w kreatorze raportu XMLP można korzystać tylko z jednego źródła danych. Na przykład z jednego zapytania SQL lub z jednego pliku XML. W praktyce jest to duże ograniczenie dla tworzenia złożonych raportów,
- w *Reports Builder* bardzo łatwo (przy użyciu myszki) grupować dane. Jest to bardzo użyteczna i prosta w użyciu cecha. W XMLP grupowanie danych pochodzących z pliku XML może być pracochłonne,
- w *Reports Builder* zawarto bardzo dużo często wykorzystywanych w raportach gotowych do użycia elementów typu suma, % całości, średnia, itp. W XMLP takich „ułatwień” nie ma,
- raporty w XMLP bazują głównie na tabelach w plikach RTF. Nie ma w nich łatwej możliwości uzyskania efektu „rozklejenia od siebie” brzegów komórek jak pokazano przykładowo na Rysunku 12 (pola od F_3 do F_6) i Rysunku 14,
- raporty w XMLP bazują głównie na tabelach w plikach RTF. Nie ma w nich możliwości uzyskania efektu automatycznego dopasowywania wielkości komórek tablicy do np. szerokości danych,
- konwersja wykorzystywanych raportów zapisanych w formacie RDF i/lub JSP do XMLP nie wydaje się w najbliższym czasie możliwa,
- w XMLP (przynajmniej na razie) brak jest zupełnie obsługi błędów. Raport albo wykona się poprawnie, albo uzyskamy informację, że z powodu błędu (nie jest on opisany) nie wykonał się.

3.3. Zalety pakietu XMLP

Pakiet XMLP jest z pewnością dużo prostszy w obsłudze niż moduł *Reports*. Skutkuje to oczywiście pewnymi ograniczeniami, jednak wydaje się, że tak proste w obsłudze narzędzia mają racje bytu. Szczególnie dobrze przyjęta przez użytkowników będzie z pewnością możliwość budowy szablonu raportu w zwykłym edytorze tekstowym. Nawet jeśli techniczne szczegóły budowy raportów (konieczność używania znaczników XSL / XSL-FO) nie będą dostępne dla szerokiego grona, to edycja np. wyglądu gotowego raportu będzie już zadaniem bardzo prostym.

Ważna jest również dostępność środowiska dewelopera w postaci aplikacji internetowej, co nie wymaga instalacji oprogramowania na komputerach programistów.

4. Wnioski

Moduł *Reports* ma ugruntowaną pozycję na rynku i jest efektywnym narzędziem do tworzenia raportów. Na dzień dzisiejszy pakiet XMLP nie jest oficjalnie promowany przez firmę Oracle na następcę modułu *Reports*. Nieobecność jednak modułów związanych z raportami w wersjach 10.1.3 zarówno OAS-a jak i JDeveloper jest zagadkowa. Pakiet XMLP, w obecnej postaci, wydaje się być przeznaczonym do szybkiego tworzenia prostych raportów. Bardziej złożone raporty na chwilę obecną chyba łatwiej jest budować w środowisku *Reports Builder*.

Bibliografia

- [Ora1] Oracle Application Server 10g Release 2 (10.1.2.0.2)
Forms and Reports Services, Oracle Application Server Reports Services Publishing Reports to the Web 10g Release 2 (10.1.2), B14048-02
- [Ora2] Oracle® Reports Building Reports 10g Release 2 (10.1.2), B13895-01
- [Ora3] Oracle® Application Server Containers for J2EE User's Guide 10g Release 2 (10.1.2), B14011-02
- [Rei06] Reimschuessel-Wąs B.: Oracle XML Publisher - nowe narzędzie do raportowania Oracle, Materiały konferencyjne PLOUG2006, Zakopane, październik, 2006
- [Tra00] Traczyk T.: Język XSL, Materiały konferencyjne PLOUG2000, Zakopane, październik, 2000, ISSN 1641-2117
- [Tra02] Traczyk T.: Obiekty formatujące w języku XSL, Materiały konferencyjne PLOUG2002, Zakopane, październik, 2002, ISSN 1641-2117
- [Tra06] Traczyk T.: Raportowanie w XML dla zwykłych ludzi - Oracle XML Publisher, Materiały konferencyjne PLOUG2006, Zakopane, październik, 2006

