Architektura zorientowana na usługi.
Ćwiczenia laboratoryjne

SKRYPT

Krzysztof Rzecki

Wydział Fizyki, Matematyki i Informatyki
Politechniki Krakowskiej
Kraków 2011
Spis treści

Wprowadzenie ........................................................................................................................................ 6

1. Serwer GlassFish .................................................................................................................................. 8
   1.1. Tematyka laboratorium .................................................................................................................. 8
   1.2. Cel laboratorium ............................................................................................................................. 8
   1.3. Zagadnienia do przygotowania ...................................................................................................... 9
   1.4. Opis laboratorium ............................................................................................................................ 9
       1.4.1. Instalacja serwera GlassFish .................................................................................................. 9
       1.4.2. Weryfikacja instalacji ............................................................................................................. 10
       1.4.3. Deploy ..................................................................................................................................... 11
       1.4.4. Undeploy .................................................................................................................................. 11
       1.4.5. Redeploy ............................................................................................................................... 12
       1.4.6. Wywołanie aplikacji z przeglądarki internetowej ................................................................. 12
       1.4.7. Narzędzie asadmin ................................................................................................................ 13
       1.4.8. Zarządzanie domenami .......................................................................................................... 13
       1.4.9. Konfiguracja domeny ............................................................................................................ 16
       1.4.10. Pula wątków ........................................................................................................................ 16
       1.4.11. Administracja łącznością internetową .................................................................................. 17
       1.4.12. Monitoring ............................................................................................................................ 18
       1.4.13. Logowanie ............................................................................................................................ 20
   1.5. Przebieg laboratorium ...................................................................................................................... 21
       1.5.1. Zadanie 1. Instalacja serwera GlassFish w wersji 3.1.1 ....................................................... 21
       1.5.2. Zadanie 2. Tworzenie domeny, (un)deploying aplikacji ...................................................... 21
       1.5.3. Zadanie 3. Testowanie wydajności serwera .............................................................................. 22
       1.5.4. Zadanie 4. Monitorowanie serwera ......................................................................................... 22

2. NetBeans .................................................................................................................................................. 23
   2.1. Tematyka laboratorium ................................................................................................................... 23
   2.2. Cel laboratorium ............................................................................................................................. 24
   2.3. Zagadnienia do przygotowania ...................................................................................................... 24
   2.4. Opis laboratorium ............................................................................................................................ 24
       2.4.1. Instalacja NetBeans ................................................................................................................ 24
       2.4.2. Tworzenie nowych projektów aplikacji ............................................................................... 25
       2.4.3. Importowanie bibliotek ......................................................................................................... 26
       2.4.4. Profiler .................................................................................................................................... 26
       2.4.5. SVN w NetBeans IDE .......................................................................................................... 29
       2.4.6. Integracja IDE z GlassFish v3 ............................................................................................ 32
   2.5. Przebieg laboratorium ...................................................................................................................... 32
       2.5.1. Zadanie 1. Zarządzanie projektami .................................................................................. 32
       2.5.2. Zadanie 2. Użycie Profilera ............................................................................................. 33

3. Servlety .................................................................................................................................................... 34
   3.1. Tematyka laboratorium .................................................................................................................. 34
   3.2. Zagadnienia do przygotowania ..................................................................................................... 34
<table>
<thead>
<tr>
<th>Spis treści</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3. Opis laboratorium .................................................................</td>
<td>35</td>
</tr>
<tr>
<td>3.3.1. Cykl życia Servletu ..........................................................</td>
<td>35</td>
</tr>
<tr>
<td>3.3.2. Podstawowe API Servletów ..................................................</td>
<td>36</td>
</tr>
<tr>
<td>3.3.3. Kompilacja i instalacja servletu .........................................</td>
<td>37</td>
</tr>
<tr>
<td>3.3.4. Servlet HTTP ............................................................................</td>
<td>38</td>
</tr>
<tr>
<td>3.3.5. Aplikacje webowe .....................................................................</td>
<td>39</td>
</tr>
<tr>
<td>3.3.6. Kontekst Servletu .....................................................................</td>
<td>40</td>
</tr>
<tr>
<td>3.3.7. Żądanie servletu .......................................................................</td>
<td>42</td>
</tr>
<tr>
<td>3.3.8. Odpowiedź servletu ..................................................................</td>
<td>42</td>
</tr>
<tr>
<td>3.3.9. Request Dispatching ..............................................................</td>
<td>43</td>
</tr>
<tr>
<td>3.3.10. Określanie kodów statusu ....................................................</td>
<td>43</td>
</tr>
<tr>
<td>3.3.11. Filtrowanie żądań i odpowiedzi ...........................................</td>
<td>44</td>
</tr>
<tr>
<td>3.3.12. Cookies ..................................................................................</td>
<td>46</td>
</tr>
<tr>
<td>3.3.13. Śledzenie Sesji ......................................................................</td>
<td>47</td>
</tr>
<tr>
<td>3.4. Przebieg laboratorium ..................................................................</td>
<td>48</td>
</tr>
<tr>
<td>3.4.1. Zadanie 1. Wykorzystanie mechanizmu cookies ......................</td>
<td>48</td>
</tr>
<tr>
<td>3.4.2. Zadanie 2. Dostęp do parametrów żądania servletu .................</td>
<td>48</td>
</tr>
<tr>
<td>3.4.3. Zadanie 3. Servlet z prostą autoryzacją dostępu ....................</td>
<td>48</td>
</tr>
<tr>
<td>4. Java Server Pages ............................................................................</td>
<td>49</td>
</tr>
<tr>
<td>4.1. Tematyka laboratorium .............................................................</td>
<td>49</td>
</tr>
<tr>
<td>4.2. Cel laboratorium ..........................................................................</td>
<td>49</td>
</tr>
<tr>
<td>4.3. Zagadnienia do przygotowania ..................................................</td>
<td>50</td>
</tr>
<tr>
<td>4.4. Opis laboratorium .......................................................................</td>
<td>50</td>
</tr>
<tr>
<td>4.4.1. Wstęp do JSP ...........................................................................</td>
<td>50</td>
</tr>
<tr>
<td>4.4.2. Alternatywne rozwiązania ......................................................</td>
<td>52</td>
</tr>
<tr>
<td>4.4.3. Problem z Servletami ..............................................................</td>
<td>53</td>
</tr>
<tr>
<td>4.4.4. Anatomia strony JSP ...............................................................</td>
<td>53</td>
</tr>
<tr>
<td>4.4.5. Przetwarzanie strony JSP ........................................................</td>
<td>54</td>
</tr>
<tr>
<td>4.4.6. Tworzenie i zarządzanie stronami JSP ....................................</td>
<td>55</td>
</tr>
<tr>
<td>4.4.7. Architektura i Używanie biblioteki JSTL ................................</td>
<td>56</td>
</tr>
<tr>
<td>4.4.8. Współpraca JSP i Servletów ..................................................</td>
<td>57</td>
</tr>
<tr>
<td>4.4.9. Prekompilacja strony JSP .......................................................</td>
<td>58</td>
</tr>
<tr>
<td>4.5. Przebieg laboratorium ..................................................................</td>
<td>60</td>
</tr>
<tr>
<td>4.5.1. Zadanie 1. Elementy JSP w akcji ............................................</td>
<td>60</td>
</tr>
<tr>
<td>4.5.2. Zadanie 2. Własne elementy akcji ..........................................</td>
<td>60</td>
</tr>
<tr>
<td>4.5.3. Zadanie 3. JSP i servlety .........................................................</td>
<td>60</td>
</tr>
<tr>
<td>4.6. Opracowanie i sprawozdanie ......................................................</td>
<td>61</td>
</tr>
<tr>
<td>5. XML i SOAP ....................................................................................</td>
<td>62</td>
</tr>
<tr>
<td>5.1. Tematyka laboratorium .............................................................</td>
<td>62</td>
</tr>
<tr>
<td>5.2. Cel laboratorium ..........................................................................</td>
<td>62</td>
</tr>
<tr>
<td>5.3. Zagadnienia do przygotowania ..................................................</td>
<td>63</td>
</tr>
<tr>
<td>5.4. Opis laboratorium .......................................................................</td>
<td>63</td>
</tr>
<tr>
<td>5.4.1. Język XML ................................................................................</td>
<td>63</td>
</tr>
<tr>
<td>5.4.2. Schematy XML .........................................................................</td>
<td>65</td>
</tr>
</tbody>
</table>
5.4.3. Protokół SOAP .................................................................66
5.4.4. Język XML w Java ..........................................................67
5.5. Przebieg laboratorium ..........................................................72
  5.5.1. Zadanie 1. Zapis pliku XML w Java ....................................72
  5.5.2. Zadanie 2. Odczyt pliku XML z waliadcją w Java ..........73
  5.5.3. Zadanie 3. Biblioteka JAXB i obiekty POJO ....................73
6. Web service, WSDL i UDDI ......................................................74
  6.1. Tematyka laboratorium .........................................................74
  6.2. Cel laboratorium .................................................................74
  6.3. Zagadnienia do przygotowania ............................................75
  6.4. Opis laboratorium ...............................................................75
    6.4.1. SOA wiadomości wstępne ..............................................76
    6.4.2. Usługi sieciowe Web Services ........................................76
    6.4.3. WSDL – Web Services Description Language ................77
    6.4.4. Praktyka – Web Services oraz WSDL ..............................77
    6.4.5. Universal Description Discovery and Integration .............80
  6.5. Przebieg laboratorium ..........................................................85
    6.5.1. Zadanie 1. Implementacja usługi CalcWS ......................85
    6.5.2. Zadanie 2. Stworzenie klienta ........................................86
    6.5.3. Zadanie 3. Wdrożenie usługi do serwera rejestrów ..........86
7. Szyka ESB ..............................................................................88
  7.1. Tematyka laboratorium .........................................................88
  7.2. Zagadnienia do przygotowania ............................................88
  7.3. Opis laboratorium ...............................................................88
    7.3.1. Architektura klasyczna ..................................................89
    7.3.2. Architektura ESB .........................................................90
    7.3.3. JBI .........................................................................90
    7.3.4. OpenESB ................................................................91
    7.3.5. NetBeans i OpenESB ....................................................93
    7.3.6. BPEL Hello World ......................................................93
    7.3.7. Klient aplikacji HelloCA ..............................................98
  7.4. Przebieg laboratorium ..........................................................98
  7.5. Zadanie 1. Testowanie komponentu sun-file-binding ...............98
  7.6. Zadanie 2. Testowanie komponentu sun-mail-binding ..........99
8. Język BPEL ............................................................................100
  8.1. Tematyka laboratorium .......................................................100
  8.2. Cel laboratorium .................................................................100
  8.3. Zagadnienia do przygotowania ............................................101
  8.4. Opis laboratorium ...............................................................101
    8.4.1. Orkiestracja i choreografia usług ..................................101
    8.4.2. Proces biznesowy ........................................................102
    8.4.3. NetBeans i graficzne edytory ........................................102
    8.4.4. Advanced Hello World w BPEL .................................104
    8.4.5. Integracja z usługami Web Services .........................107
    8.4.6. Zaawansowane konstrukcje BPEL .............................108
8.5. Przebieg laboratorium ................................................................. 112
  8.5.1. Przygotowanie laboratorium ............................................. 112
  8.5.2. Zadanie 1. Prowizja .......................................................... 113
  8.5.3. Zadanie 2. Prototyp biura podróży ................................. 113
Wprowadzenie

Pierwotne metody informatyczne stosowane w ubiegłym stuleciu do realizacji ciągów operacji biznesowych prowadzących do osiągnięcia założonego celu, opierały się o przenoszenie danych na tradycyjnym nośniku informacji – na papierze. Przykładem może być zakład produkujący małe elementy metalowe, do którego zlecenie na nowy element przysyłane było najczęściej w formie papierowej, dokument rejestrowany w rejestrze dokumentów przechodzących trafiał w obieg (zwykle zmieniając swoją zawartość po drodze) od kosztorysanta, przez projektanta, hałę produkcyjną, kontrolę jakości, magazyn aż po sprzedaż. Cały proces realizacji zamówienia kontrolowany był tak, aby zapewnić prawidłową, zgodną z dokumentacją realizację zamówienia.

Pierwsza informatyzacja procesów biznesowych opierała się o przetworzenie tej informacji z wersji papierowej na elektroniczną. Polegało to na indeksacji dokumentów, ich gromadzeniu, przetwarzaniu, archiwizowaniu, itp. z wykorzystaniem komputera. Ewentualna wędrówka informacji po kolejnych miejscach realizacji operacji biznesowych polegała na przenoszeniu jej z użyciem nośników cyfrowych (karty, taśmy, dyskietki) lub znów na papierze w formie wydruku.

Dopiero technologie sieciowe dały możliwość przesyłania informacji skojarzonej z realizacją danego procesu biznesowego z pominięciem nośników wymagających bezpośredniego zaangażowania człowieka. Programowanie usług sieciowych pozwoliło tworzyć oprogramowanie umożliwiające wędrówkę informacji między kolejnymi systemami informatycznymi wchodzącymi w skład ścieżki realizacji procesu biznesowego. Usługi sieciowe (ang. network services) znalazły zastosowanie w technologiach na wyższym poziomie abstrakcji, jak np.: w programowaniu rozproszonym, obliczeniach równoległych, technologiach gridowych.

Opracowanie wysoko abstrakcyjnej koncepcji architektury zorientowanej na usługi (SOA, ang. Service-Oriented Architecture) wniosło reguły projektowania systemów, w których usługa to niezależne (od innych usług, platformy, technologii wykonania, itp.) oprogramowanie mogące wejść w skład złożonego procesu biznesowego. Z usługi SOA można korzystać komunikując się z nią za pomocą bardzo elastycznego definiowalnego protokołu komunikacyjnego. Do tego celu usługa udostępnia interfejs, który mówi tylko co usługa może zrealizować i z założenia nie informuje jak wykona złożone zlecenie.

Obecnie rozwijane są technologie na jeszcze wyższym poziomie abstrakcji, tj. chmury obliczeniowe (ang. cloud computing). W tym przypadku dostarczanie usług realizowane jest z obszaru teleinformatycznego nieidentyfikowanego bezpośrednio ze sprzętem. Chmurę może być każdy z poziomów architektury, od sprzętu, przez infrastrukturę, platformę aż po usługę.

Niniejszy skrypt przedstawia inżynieryjne wprowadzenie do architektury zorientowanej na usługi. Skrypt składa się z ośmiu następujących po sobie ćwiczeń laboratoryjnych, począwszy od tematu serwera aplikacji, środowiska programistycznego, przez podstawowe technologie programistyczne aż po interfejs i protokół oraz elementy architektury. Każde ćwiczenie składa się z przedstawienia tematyki laboratorium, zagadnień do przygotowania przed przystąpieniem do ćwiczenia, z opisu laboratorium (które zajmuje najobszerniejszą część ćwiczenia) oraz zadań do samodzielnjej realizacji.
Wprowadzenie

Realizacja kolejnych laboratoriów polega na wykonaniu wszystkich zadań podanych w drugiej części danej instrukcji. Wynikiem wykonania powinno być sprawozdanie w formie wydruku papierowego, które zawiera:

- opis metodyki realizacji zadań (system operacyjny, język programowania, biblioteki, itp.),
- algorytmy wykorzystane w zadaniach (zwłaszcza, jeśli zastosowane zostały rozwiązania nietypowe),
- opisy napisanych programów wraz z opcjami,
- uwagi oceniające ćwiczenie: trudne/łatwe, nie/wymagające wcześniejszej znajomości zagadnień (wymienić jakich),
- wskazówki dotyczące ewentualnej poprawy instrukcji celem lepszego zrozumienia sensu oraz treści zadań.

Skrypt powstał przy współpracy z mgr inż. Michałem Niedźwieckim, doktorantem AGH, którego nieoceniona pomoc w zakresie wsparcia teoretycznego i praktycznego pozwoliła na ukształtowanie ostatecznej postaci ćwiczeń.

Dzięki współpracy studentów Politechniki Krakowskiej, a w szczególności byli to Michał Sierpień oraz Kamil Wójcik, skrypt zawiera informacje i zadania sprawdzone praktycznie. Znaczący wkład w analizę spójności, weryfikację wykonalności zadań oraz korektę tekstu wnieśli studenci Marek Wojtaszek i Mateusz Żurek.

Mimo wszelkich starań, opracowanie może zawierać błędy polegające na zróżnicowanej interpretacji zagadnień dotyczących technologii SOA, której dynamiczny i stały rozwój stale przynosi zmiany.
1. Serwer GlassFish

W tym ćwiczeniu w pierwszej kolejności zapoznamy się z serwerem GlassFish, jego instalacją, konfiguracją i administracją. W drugiej części instrukcji znajdują się zadania z tego zakresu przeznaczone do samodzielnej realizacji.

1.1. Tematyka laboratorium

Serwerowe aplikacje Java można uruchamiać m.in. za pomocą serwera aplikacji GlassFish. Serwer aplikacji umożliwia efektywne zarządzanie i wykorzystywanie istniejących komponentów przez uruchamianą aplikację, co jest podstawową zaletą w porównaniu z tradycyjnym systemem operacyjnym. Serwer aplikacji pozwala na zarządzanie komponentami oraz na komunikację między nimi. Serwer aplikacji zapewnia także środowisko do uruchamiania aplikacji oraz udostępnia je w sieci dając możliwość połączeń z komponentami przez sieć Internet.

Inne popularne serwery aplikacji to między innymi:
- Apache Geronimo (Apache Software Foundation),
- Glassfish Application Server (Oracle Corporation),
- WebSphere Application Server and WebSphere Application Server Community Edition (IBM),
- JBoss (Red Hat),
- Jetty (Eclipse Foundation),
- SAP Netweaver Application Server (ABAP/Java) (SAP),
- Oracle GlassFish Enterprise Server (wcześniej Sun GlassFish Enterprise Server bazujący na GlassFish Application Server) (Oracle Corporation).

1.2. Cel laboratorium

Celem laboratorium jest poznanie technik konfiguracji i administracji serwera GlassFish. Techniki te są bazą w tworzeniu aplikacji webowych w technologii Java EE. Podczas realizacji tego laboratorium zapoznasz się z:
- administracją z poziomu konsoli oraz Panelu administracyjnego,
- konfiguracją serwera,
- uruchomieniem i testowaniem serwera.
1.3. Zagadnienia do przygotowania

Przed przystąpieniem do realizacji laboratorium należy zapoznać się z zagadnieniami dotyczącymi:

- komponent, moduł, kontener
- podstawy technologii Java EE
- znajomość powłoki basha
- znajomość Java SE

Podstawowa literatura:
[1] Java EE 6 with Glassfish Application Server
[2] Java Enterprise In A Nutshell
[3] Oracle® GlassFish Server 3.1.1 Administration Guide
[5] Oracle® GlassFish Server 3.1.1 Application Deployment Guide

1.4. Opis laboratorium

Zakres laboratorium koncentruje się na zapoznaniu się z podstawową obsługą serwera GlassFish, który będzie wykorzystywany w kolejnych laboratoriach. W celu wykonania ćwiczeń praktycznych przydatne będą przedstawione w tym podrozdziale informacje teoretyczne.

1.4.1. Instalacja serwera GlassFish


Serwer GlassFish 3.1.1 wymaga środowiska JDK w wersji co najmniej 6. Jeśli nie ma zainstalowanego JDK należy je pobrać i zainstalować. Po instalacji należy upewnić się, że zmienne środowiskowe JAVA_HOME i $JAVA_HOME/bin są ustawione na katalog w którym został zainstalowany JDK, w przeciwnym wypadku należy je ustawić poprzez dodanie do pliku /etc/profile linijek:

```
export JAVA_HOME=/usr/lib/jvm/java-6-sun-1.6.0.26/
export PATH=$PATH:$JAVA_HOME/bin
```

Z powodów bezpieczeństwa, instalacji nie należy wykonywać na koncie roota. Do tego celu zostanie utworzone nowe konto glassfish (operacja wymaga uprawnień roota).

```
$ sudo adduser glassfish
```
Poniżej znajdują się ważniejsze kroki instalacji, a pominięte okna można zaakceptować bez wprowadzania zmian:

1. Przed rozpoczęciem instalacji należy się zalogować na konto glassfish.
2. Nadaj plikowi instalacyjnemu prawa do wykonywania (jeśli trzeba):
   ```bash
   $ chmod +x nazwa_pliku
   ```
3. Uruchom instalator wpisując w linii komend:
   ```bash
   $ ./nazwa_pliku
   ```
4. Uruchom się graficzny instalator.
5. W oknie Installation Directory wybieramy katalog w którym zostanie zainstalowany GlassFish. Domyślnie `/home/<user>/glassfish3`.
7. W oknie Administration Settings ustawiamy numery portów – należy zadbać o to, aby wskazywane porty były nie używane:
   - **Admin Port** – port z którego będziemy się łączyć do panelu administacyjnego (domyślnie 4848).
   - **Http Port** – port który będzie wykorzystywany do komunikacji z serwerem przy użyciu HTTP (domyślnie 8080).
   - **Username/Password** – login/hasło do administracji serwerem (domyślnie admin/null).

   W poniżej tabeli znajdują się standardowe ustawienia dotyczące położenia poszczególnych części serwera GlassFish w systemie plików po zainstalowaniu. W razie konieczności należy dodać podane aliasy do powłoki, z której korzystamy.

<table>
<thead>
<tr>
<th>Alias</th>
<th>Opis</th>
</tr>
</thead>
<tbody>
<tr>
<td>as-install</td>
<td>reprezentuje domyślną lokalizację folderu instalacji serwera GlassFish</td>
</tr>
<tr>
<td>as-install-parent</td>
<td>reprezentuje folder nadrzędny dla as-install</td>
</tr>
<tr>
<td>domain-root-dir</td>
<td>reprezentuje domyślną lokalizację domen</td>
</tr>
<tr>
<td>domain-dir</td>
<td>reprezentuje położenie plików konfiguracyjnych dla konkretnej domeny</td>
</tr>
</tbody>
</table>

### 1.4.2. Weryfikacja instalacji

Weryfikacja instalacji polega na sprawdzeniu, czy serwer można uruchomić:

1. W linii poleceń przejść do katalogu z zainstalowanym oprogramowaniem:
   ```bash
   $ cd as-install/bin
   ```
2. Wykonać komendę:
   ```bash
   $ ./asadmin start-domain domain1
   ```
3. Powinien pojawić się komunikat podobny do poniższego:
   ```
   Waiting for DAS to start ......................
   Started domain: domain1
   Domain location: /home/<user>/glassfish3/ \
   glassfish/domains/domain1
   Logfile:/home/<user>/glassfish3/glassfish/ \
   domains/pkdomain/logs/server.log
   ```
1.4.3. Deploy

Słowo „deploy” oznacza umieszczenie aplikacji (jej komponentów) w serwerze aplikacji, co odpowiada jej instalacji. Proces ten nazywać dalej będziemy „deployingiem”. W jego trakcie odpowiednie narzędzie rozwpakowuje pliki archiwum (war, jar) do otwartej struktury katalogowej z plikami klas i jsp gotowymi do użycia.

Zasadniczo wyróżniamy trzy sposoby na deploying aplikacji tj. przez:
- panel administracyjny w przeglądarce internetowej,
- z użyciem narzędzia asadmin,
- z wykorzystaniem opcji autodeploy.

Rozwiązanie z panelem administracyjnym polega na przejściu na adres serwera na porcie 4848: http://localhost:4848/. Po zalogowaniu, przechodzimy do sekcji Applications i wybieramy opcję „Deploy”. Kolejno w polu „Packaged File to Be Uploaded to the Server” wybieramy plik WAR (web application archive) znajdujący się w folderze ze źródłami. Po wykonaniu uploadu powinniśmy zobaczyć tabele, a w niej naszą aplikację.

Deploy z użyciem narzędzia asadmin polega na uruchomieniu:
$ as-install/bin/asadmin
Polecenie, które wykona deploy aplikacji to:
$ deploy <ścieżka do pliku>
Po udanej instalacji otrzymamy informację:
Application deployed successfully with name ... 
Command deploy executed successfully.

Ostatni sposób deployingu aplikacji polega na umieszczeniu pliku WAR w folderze domain-root-dir/domain1/autodeploy. Serwer, który cyklicznie sprawdza zawartość tego katalogu, sam dokona deployingu aplikacji.

Podstawowy URL zdeployowanej aplikacji to: http://localhost:8080/<aplikacja>

1.4.4. Undeploy

Są trzy sposoby na undeploying, czyli usunięcie aplikacji z serwera poprzez:
- panel administracyjny w przeglądarce internetowej,
- z użyciem narzędzia asadmin,
- z wykorzystaniem opcji autodeploy.

Narzędzie asadmin uruchamiane jest tak samo jak w przypadku deployingu, a komenda wyładowująca aplikację ma składnię:

$ undeploy <nazwa komponentu>

Operacja zakończona pozytywnie zwróci komunikat:

Command undeploy executed successfully.

Korzystanie z automatycznego undeploingu polega na usunięciu wcześniej skopiowanego pliku WAR z katalogu domain-root-dir/domain1/autodeploy. Serwer sam dokona undeploingu aplikacji.

1.4.5. Redeploy

Ponowne załadowanie aplikacji, bez konieczności jej wyładowania i załadowania, możliwe jest przy pomocy funkcjonalności redeploy. Redeploy z użyciem panelu administracyjnego w przeglądarce internetowej oraz z wykorzystaniem opcji autodeploy wykonuje się analogicznie do deploy i undeploy. Redeploy z narzędzia asadmin wykonuje się w następujący sposób:

asadmin> redeploy --name <app>
Application deployed successfully with name <app>.
Command redeploy executed successfully

1.4.6. Wywołanie aplikacji z przeglądarki internetowej

Po umieszczeniu servletu na serwerze można go wywołać używając do tego celu przeglądarki lub zrobić to w pliku HTML lub JSP. Format URL odnoszącego się do servletu jest przedstawiony poniżej:

http://server:port/context-root/servlet-mapping?name=value

Objaśnienie poszczególnych napisów przedstawia poniższa tabela:

<table>
<thead>
<tr>
<th>Element URL</th>
<th>Opis</th>
</tr>
</thead>
<tbody>
<tr>
<td>server:port</td>
<td>Adres IP (lub nazwa hosta) oraz opcjonalnie numer portu. Jest to domyślny adres modułu web serwera wirtualnego i nie wymaga podania context-root ani servlet-name (chyba, że wymagane jest użycie parametrów dla żądania).</td>
</tr>
<tr>
<td>context-root</td>
<td>Dla aplikacji context-root jest zdefiniowany w elemencie o tej samej nazwie w plikach application.xml, sun-application.xml, lub sun-web.xml. Dla osobno zdeployowanych modułów web, context-root jest określany podczas deployingu. Zarówno dla aplikacji jaki i osobno zdeployowanych modułów web, domyślną wartością context-root jest nazwa WAR'u bez rozszerzenia (.war).</td>
</tr>
<tr>
<td>servlet-mapping</td>
<td>Odpowiednik servlet-mapping w pliku web.xml</td>
</tr>
<tr>
<td>?name=value</td>
<td>Opcjonalne parametry żądania w zapisie standardowym.</td>
</tr>
</tbody>
</table>
1.4.7. Narzędzie asadmin

Narzędzie to służy do administracji serwerem i może zastąpić graficzny panel administracyjny dostępny z poziomu przeglądarki. Niektóre opcje dostępne są tylko za pomocą narzędzia asadmin. Domyślnie narzędzie to jest zlokalizowane w katalogu as-install/bin.

Składnia polecenia:

```
asadmin [asadmin-util-options] \n    [subcommand [subcommand-options] [operands]]
```

Narzędzie asadmin może pracować w dwóch trybach: Single Mode i Multi Mode. W trybie Single Mode wymagane jest wpisywanie /asadmin dla każdego polecenia które chcemy wykonać, na przykład:

```
./asadmin list-domains
```

Po wykonaniu polecenia sterowanie powracaje do konsoli powłoki systemowej. W tym trybie, jeśli używane są jakiekolwiek opcje, muszą one być podawane dla każdego polecenia oddzielnie. W przypadku gdy opcje są takie same dla kolejnych poleceń, należy użyć trybu Multi Mode.

W trybie Multi Mode uruchomienie narzędzia asadmin powoduje rozpoczęcie sesji. Kolejne polecenia będą uruchamiane z opcjami, z którymi zostało uruchomione narzędzie asadmin. Aby pracować w tym trybie należy: uruchomić asadmin bez żadnych podpoleceń lub użyć podpolecenia multimode. Po przejściu w tryb multimode zmieniony zostanie znak zachęty na tryb konsoli asadmin. Aby wyjść z trybu multimode wystarczy podać polecenie exit (lub quit) albo wcisnąć sekwencję ctrl+d.

Pomoc dotycząca narzędzia asadmin jest dostępna poprzez wydanie komendy help <nazwa_polecenia>. Można tym sposobem dostać się do tekstu pomocy zarówno prostych komend jak list, jak i do złożonych komend jak list-containers.

Przykład:

```
asadmin> help list-domains
```

1.4.8. Zarządzanie domenami

Domeny to grupy instancji, które są zarządzane wspólnie. Domena zapewnia prekonfigurowane środowisko do uruchamiania aplikacji użytkownika. Domeny zapewniają granice administracji instancjami oraz podstawową strukturę zabezpieczeń, za pomocą czego różni administratorzy mogą zarządzać określoną grupą instancji. Poprzez grupowanie różnych instancji na serwerze w rozdzielne domeny, różne organizacje mogą dzielić jeden serwer aplikacji.

Domena posiada własną konfigurację, logi oraz niezależne od innych domen obszary deployingu. Zmiany w jednej domenie nie wpływają na inne domeny na tym samym serwerze. Instalator serwera GlassFish domyślnie tworzy domenę nazwaną „domain1” jak i DAS (ang. domain administration server).

DAS jest to specjalnie zaprojektowana instancja, która zapewnia uwierzytelnienie administratorom, akceptuje żądania z panelu administratorycznego i komunikuje się z instancjami konkretnych domen w celu obsługi żądania. DAS czasami odnosi się do serwera domyślnego, ponieważ jest to jedyna instancja serwera tworzona w czasie instalacji.
Panel administracyjny, dostępny poprzez przeglądarkę, komunikuje się z określonym DAS-em w celu zarządzania domenami z nim powiązanymi. Każda sesja logowania do panelu administracyjnego pozwala na konfigurację i zarządzanie jedną domeną. Jeśli na serwerze jest więcej domen, dla każdej domeny potrzebna jest osobna sesja logowania do panelu administracyjnego.

Aby stworzyć nową domenę należy w linii komend wydać polecenie create-domain. Każdy, kto może wykonywać polecenia z poziomu asadmin, może tworzyć nowe domeny. Domyślnie domeny tworzone są w folderze glassfish/domains, jednak przy wydawaniu polecenia można podać dowolną lokalizację dla nowej domeny. Należy zadbać, aby porty których będzie używać nowa domena nie były wykorzystywane przez inne usługi.

Domyślnie wartości portów są ustawione na:

<table>
<thead>
<tr>
<th>Service</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admin</td>
<td>4848</td>
</tr>
<tr>
<td>HTTP</td>
<td>8080</td>
</tr>
<tr>
<td>Java Messaging System (JMS)</td>
<td>7676</td>
</tr>
<tr>
<td>Internet Inter-ORB Protocol (IIOP)</td>
<td>3700</td>
</tr>
<tr>
<td>Secure HTTP (HTTPS)</td>
<td>8181</td>
</tr>
<tr>
<td>Secure IIOP</td>
<td>3820</td>
</tr>
<tr>
<td>Mutual Authorization IIOP</td>
<td>3920</td>
</tr>
<tr>
<td>Java Management Extensions</td>
<td>8686</td>
</tr>
<tr>
<td>Administration</td>
<td></td>
</tr>
</tbody>
</table>

Przy tworzeniu nowych domen istnieje możliwość wykorzystania opcji portbase. Służy ona do podania wartości portu bazowego w stosunku do którego, z odpowiednim przesunięciem będą ustawiane wartości portów dla kolejnych usług z których korzysta nowa domena.

<table>
<thead>
<tr>
<th>Tabela z przesunięciami</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>Port</td>
</tr>
<tr>
<td>Admin</td>
<td>portbase + 48</td>
</tr>
<tr>
<td>HTTP</td>
<td>portbase + 80</td>
</tr>
<tr>
<td>Java Messaging System (JMS)</td>
<td>portbase + 76</td>
</tr>
<tr>
<td>Internet Inter-ORB Protocol (IIOP)</td>
<td>portbase + 37</td>
</tr>
<tr>
<td>Secure HTTP (HTTPS)</td>
<td>portbase + 81</td>
</tr>
<tr>
<td>Secure IIOP</td>
<td>portbase + 38</td>
</tr>
<tr>
<td>Mutual Authorization IIOP</td>
<td>portbase + 39</td>
</tr>
<tr>
<td>Java Management Extensions</td>
<td>portbase + 86</td>
</tr>
<tr>
<td>Administration</td>
<td></td>
</tr>
</tbody>
</table>

Dobrą praktyką przy tworzeniu kolejnych domen z wykorzystaniem portbase, aby nie zachodził konflikt używanych portów, jest ustawianie wartości opcji portbase na 8000 i w przy tworzeniu kolejnych domen zwiększanie tej wartości o 1000.

Przykład:

```
   asadmin> create-domain --portbase 9000 \
   --domaindir /home/user domainA
```

Polecenie utworzy w lokalizacji /home/user domenę o nazwie domainA, z portem administracyjnym 9048, portem HTTP 9080 itd. Przy tworzeniu domen wymagane jest ustawienie loginu i hasła dla nowej domeny.
Aby wyświetlić utworzone wcześniej domeny i ich status należy posłużyć się poleceniem list-domains:

```
asadmin > list-domains
Name: domainA Status: Not Running
Name: domainB Status: Running
Command list-domains executed successfully.
```

Jeżeli domena została stworzona w innej lokalizacji niż domyślna, należy użyć opcji --domaindir "ścieżka do katalogu".

Usuwanie domeny należy poprzedzić jej zatrzymaniem. Do usunięcia istniejącej domeny wykorzystuje się polecenie delete-domain. Jeżeli domena znajduje się w katalogu innym niż domyślny należy użyć opecji --domaindir:

```
asadmin> delete-domain domain1
Domain domain1 deleted.
Command delete-domain executed successfully.
```

Uruchamianie domeny lub całego serwera pociąga za sobą uruchomienie DAS-a. DAS będzie pracował w trybie ciągłym oczekując na żądania. Każda z domen musi zostać uruchomiona oddzielnie. Jeżeli domena znajduje się w domyślnym katalogu domen do uruchomienia domeny wykorzystujemy polecenie:

```
asadmin > start-domain <nazwa domeny>
```

Przykład:

```
asadmin> start-domain domainA
Waiting for DAS to start .............
Started domain: domainA
Domain location: /glassfish/domains/pk2domain
Log file:/glassfish/domains/domainA/logs/server.log
Admin port for the domain: 21048
Command start-domain executed successfully.
```

Zatrzymanie domeny lub serwera powoduje zamknięcie DAS-a. Podczas zatrzymywania domeny DAS przestaje akceptować nowe połączenia i czeka na zakończenie obsługi obecnych połączeń. W zależności od ilości aplikacji na serwerze oraz ich rozmiarze, proces zatrzymywania i uruchamiania domeny może trwać dłużej albo krócej. Po zatrzymaniu domeny konsola administracyjna i większość poleceń asadmin jest nieaktywna.

Do zatrzymania domeny wykorzystywane jest polecenie

```
asadmin > stop-domain <nazwa domeny>
```

Podczas restartowania najpierw następuje zatrzymanie domeny, a następnie jej uruchomienie. W tym tym przypadku DAS zachowuje się podobnie jak w opisanych wcześniej przypadkach. Do zrestartowania domeny służy polecenie:

```
asadmin > restart-domain <nazwa domeny>
```
1.4.9. Konfiguracja domeny

Ustawienia konfiguracyjne konkretnej domeny są przechowywane w pliku domain.xml w katalogu as-install/domains/domain-name/config. Plik logging.properties znajdujący się w tej lokalizacji jest to plik zawierający ustawienia poziomów logowania dla poszczególnych modułów. Pliki konfiguracyjne przechowujące zmienne środowiskowe, ustawienia asadmin itp. znajdują się w as-install/config. Zmiany niektórych ustawień wymagają ponownego uruchomienia serwera. Ustawieniami modułów można zarządzać z linii poleceń poprzez polecenia get i set. Aby zobaczyć dostępne moduły komponentów należy wpisać komendę

```bash
asadmin > list "*"
```

Do sprawdzenia wartości ustawień danego modułu należy użyć polecenia get.

```bash
asadmin > get "nazwa modułu"
```

Do ustawiania wartości ustawienia danego modułu należy użyć metody set.

```bash
asadmin > set "nazwa modułu" = wartość
```

Uwaga: Zmiany są automatycznie wprowadzane. Odradza się edytowanie plików bezpośrednio, może to powodować nieprzewidywalne skutki.

Ustawienia konfiguracji serwera i domen można zmieniać w graficznym panelu administracyjnym. Wszystkie opcje zostały obszernie wytłumaczone w pomocy panelu administracyjnego. Poniżej omówione są opcje konfiguracji serwera z poziomu graficznego panelu administracyjnego. Po zalogowaniu do panelu, localhost:4848, po prawej stronie widoczny jest panel konfiguracji.

1.4.10. Pula wątków


Polecenie do tworzenia puli to create-threadpool. Tworzenie puli wątków wymaga, aby serwer był uruchomiony przed wydaniem polecenia oraz ponownego uruchomienia serwera po wydaniu polecenia:

```bash
asadmin> create-threadpool
--maxthreadpoolsiz 100 // maksymalny rozmiar puli
--minthreadpoolsiz 20 // minimalny rozmiar puli
--idletimeout 2 // czas życia wątku
threadpool-1 // identyfikator puli
```
Uwaga: Możliwe jest zdarzenie, w którym jedna aplikacja zajmie całą pulę wątków, wtedy serwer jest zablokowany. Aby tego uniknąć należy podzielić pule wątków na kilka mniejszych.

Polecenie do listowania pul wątków (czyli wypisania zdefiniowanych pul) to list-threadpools.
Przykład:
```
asadmin> list-threadpools
threadpool-1
Command list-threadpools executed successfully
```

Polecenie do usuwania puli to delete-threadpools. Wątki w puli w momencie usuwania nie mogą obsługiwać żadnych połączeń.
Przykład:
```
asadmin> delete-threadpool threadpool-1
Command delete-threadpool executed successfully
```

1.4.11. Administracja łącznością internetową

Usługi HTTP dostarczają funkcjonalność deployingu aplikacji webowych i dostępu do aplikacji uruchomionych na serwerze przez połączenie Internet. Usługi HTTP dostarczane są przez dwa typy powiązanych obiektów: wirtualne serwery i gniazda nasłuchowe (ang. Network Listeners).

Gniazdo nasłuchowe posiada adres IP, numer portu, nazwę serwera oraz domyślny serwer wirtualny. Każdy serwer wirtualny dostarcza połączenie pomiędzy klientem i serwerem poprzez jednego lub więcej listenerów. Każde gniazdo musi mieć unikalny adres IP i numer portu. Gniazdo może nasłuchiwać na wszystkich zdefiniowanych adresach IP poprzez przypisanie mu wartości IP 0.0.0.0. Ponieważ gniazdo nasłuchowe to para liczb IP i numeru portu, można utworzyć kilka gniazd o tym samym adresie IP ale innych numerach portów albo o tym samym numerze portu ale innych adresach IP. Jednak jeżeli stworzymy gniazdo nasłuchowe używające adresu IP 0.0.0.0 oraz specyficznego portu to nie możliwe jest stworzenie gniazda na tym samym porcie z konkretnym adresem IP.

Domyślnie przy uruchamianiu serwera GlassFish tworzone są gniazda nasłuchowe:
- http-listener-1 – nie posiada włączonej ochrony,
- http-listener-2 – ma włączoną ochronę.

Domyślnie gniazda działają na portach:
- Panel administracji – 4848,
- HTTP – 8080,
- HTTPS – 8181.

Serwery wirtualne, czasami nazywane wirtualnymi hostami, to obiekty, które pozwalają na hosting wielu domen internetowych na jednej maszynie. Wszystkie serwery wirtualne korzystają z jednego adresu IP serwera. Serwery wirtualne łączą domenę internetową z serwerem GlassFish. Każdy serwer wirtualny musi zostać
zarejestrowany przy pomocy serwera DNS dla twojej sieci. Serwer wirtualny jest wirtualnym serwerem web, który udostępnia treść strony dla konkretnego URL-a.

Uwaga: Nie należy mylić domeny internetowej z domeną administracyjną serwera GlassFish. Są to zupełnie inne domeny, zwykle nie związane ze sobą.


1.4.12. Monitoring

Monitoring to proces przeglądania statystyk systemu w celu zwiększenia wydajności lub rozwiązania problemów. Usługa monitorowania pozwala nadzorować statystyki operacyjne, takie jak ilość zapytań na sekundę, średni czas odpowiedzi i przepustowość. Poprzez monitorowanie stanu różnych komponentów i usług znajdujących się na serwerze, można zidentyfikować wąskie gardła, które zmniejszają wydajność, przewidywać usterki, wykonywać analizy i zapewniać, że wszystko funkcjonuje tak jak należy. Dane zebrane podczas monitorowania mogą być również użyte do połączenia wydajności i planowania pojemności serwera. W opisywanej wersji serwera monitorowanie jest zaimplementowane w sposób modułowy, pozwala to na korzystanie z tej usługi wielu modułom klienckim. Do tych modułów zalicza się panel administracyjny, narzędzie asadmin, interfejsy AMX i REST.

Obiektem monitorowanym może być komponent, podkomponent lub usługa. GlassFish używa struktury drzewiastej do przedstawienia obiektów, które mogą być monitorowane. Ponieważ drzewo jest dynamiczne, to ulega ono zmianom podczas dodawania lub usuwania komponentów z serwera. W drzewie, obiekt monitorowany może posiadać obiekty potome (węzły) reprezentujące własności, które tak naprawdę mogą być monitorowane dla tego obiektu. Wszystkie obiekty potomne są adresowane przy pomocy znaku kropki (.) jako separatora.

Poniższe polecenia narzędzia asadmin umożliwiają monitoring usług i komponentów serwera aplikacji:

- **monitor --type** – wykorzystywane do wyświetlania podstawowych danych określonego typu monitorowalnych obiektów.
- **enable-monitoring**, **disable-monitoring** lub **get** i **set** – polecenia używane do włączenia bądź wyłączenia monitorowania.
- **list --monitor** – wyświetlenie obiektów które mogą być monitorowane przy użyciu polecenia monitor.
- **get** – wyświetla szczegółowe dane tj. atrybuty oraz ich wartości. Polecenie to używa ze znakiem wildcard (*) wyświetla wszystkie dostępne atrybuty dla wszystkich obiektów.
Domyślnie usługa monitorująca jest włączona dla serwera, ale monitorowanie poszczególnych modułów nie jest. Aby włączyć monitorowanie dla modułu należy zmienić poziom monitorowania dla tego modułu na LOW lub HIGH. Aby wyłączyć monitorowanie należy ustawić poziom na OFF.

- LOW – proste statystyki tj. liczenie utworzeń, liczenie bajtów itd.
- HIGH – proste statystyki i statystyki metod tj. liczenie metod, trwania itd.
- OFF – brak monitorowania. Brak wpływu na wydajność.


Do określenia które usługi i komponenty są aktualnie monitorowane należy użyć poniższego polecenia:

```
asadmin> get server.monitoring-service.module-monitoring-levels.*
```

Otrzymany wynik polecenia pokazuje, że monitorowanie HTTP service jest wyłączone, ale pozostałe są włączone:

```
configs.config.server-config.monitoring-service.module-monitoring-levels.web-container=HIGH
configs.config.server-config.monitoring-service.module-monitoring-levels.http-service=OFF
configs.config.server-config.monitoring-service.module-monitoring-levels.jvm=HIGH
```

Przykłady:

**Włączenie usługi monitorowania nie wpływając na monitorowanie poszczególnych modułów.**

```
asadmin> enable-monitoring
Command enable-monitoring executed successfully
```

**Włączenie monitorowania dla modułu ejb-container.**

```
asadmin> enable-monitoring --level ejb-container=HIGH
Command enable-monitoring executed successfully
```

**Włączenie monitorowania dla usługi HTTP poprzez ustawienie poziomu monitorowania na HIGH (należy zrestartować serwer aby zmiany odniosły efekt).**

```
asadmin> set server.monitoring-service.module-monitoring-levels.http-service=HIGH
Command set executed successfully.
```

Wyłączanie przebiega podobnie jak włączanie, lecz zamiast polecenia enable-monitoring należy użyć disable-monitoring. Drugim sposobem jest użycie set, gdzie poziom monitorowania należy ustawić na OFF.
Do przeglądania danych służy polecenie monitor.

Aby określić, dla którego obiektu dane mają zostać wyświetlane, należy użyć opcji --type. Parametrami mogą być min. httplistener, jvm, webmodule. Użycie tej opcji jest wymagane, inaczej zwrócony zostanie błąd. Wynik polecenia jest wyświetlany w sposób ciągły w formie tabeli. Opcja --interval służy do określenia interwału czasu (domyślnie 30s). Aby można było wyświetlić dane o obiekcie musi on zostać wcześniej skonfigurowany jak i monitorowany.

Przykład:
```
asadmin> monitor --type jvm server
```

1.4.13. Logowanie

Logowanie jest procesem, w którym serwer GlassFish przechwytuje informacje o zdarzeniach takich jak błędy konfiguracji, nieautoryzowane próby dostępu oraz usterki serwera. Plik logów domyślnie znajduje się w katalogu domeny w logs/server.log.

Format logów jest określony następująco:
```
[#|yyyy-mm-ddThh:mm:ss.SSS-Z|Log Level|ProductName-Version|LoggerName|Key Value Pairs|Message|#]
```

Znaczniki [# and #] to znaczniki początku i końca jednego wpisu logu. Pionowy znak (|) oddziela kolejne pola we wpisie. yyyy-mm-ddThh:mm:ss.SSSS-Z określa datę i godzinę o której wpis został dodany. Log Level określa docelowy poziom logowania. Dostępne są poziomy: SEVERE, WARNING, INFO (domyślny), CONFIG, FINE, FINER, i FINEST. LoggerName określa źródłowy moduł, od którego pochodzi wpis, np. javax.enterprise.system.core. Key Value Pairs odnosi się do par klucz=wartość, zazwyczaj jest to thread ID. Message jest tekstem związany z wpisem do logów.

Przykładowy wpis do logów może wyglądać podobnie do tego:
```
```

Aby zobaczyć poziomy logowania poszczególnych modułów należy użyć komendy:
```
asadmin > list-logger-levels
```

Aby zmienić poziom logów dla określonego modułu loggera należy wykorzystać polecenie:
```
asadmin > set-log-level \ <moduł loggera>=<poziom logowania>
```

Przykład:
```
asadmin> set-log-level \ javax.enterprise.system.container.web.level=FINE
Command set-log-level executed successfully.
```
1.5. Przebieg laboratorium

Drugą część instrukcji zawiera zadania do praktycznej realizacji, które demonstrują zastosowanie technik z omawianego zagadnienia.

1.5.1. Zadanie 1. Instalacja serwera GlassFish w wersji 3.1.1

Zadanie polega na zainstalowaniu serwera GlassFish 3.1.1 na stacji roboczej oraz weryfikacji instalacji poprzez uruchomienie serwera i przykładowej domeny, sprawdzenie czy serwer działa poprawnie oraz deployingu przykładowej aplikacji, znajdującej się w folderze ze źródłami, na serwer GlassFish.

Wykonanie zadania polega na wykonaniu następujących podpunktów:
1. Pobranie pliku instalacyjnego serwera oraz jego instalacja na stacji roboczej.
2. Uruchomienie domyślnej domeny oraz sprawdzenie, czy serwer został poprawnie zainstalowany.
3. Deploying przykładowej aplikacji oraz sprawdzenie, czy aplikacja działa. Przykładowa aplikacja jest serwisem, który wypisuje napis „Hello World”.

1.5.2. Zadanie 2. Tworzenie domeny, (un)deploying aplikacji

Zadanie polega na praktycznym sprawdzeniu wiedzy o zarządzaniu domenami oraz deployingu aplikacji na serwer. Należy usunąć domenę domyślną, stworzyć domenę z odpowiednimi opcjami oraz wykonać trzema metodami deployingu i undeployingu aplikacji.

Wykonanie zadania polega na wykonaniu następujących podpunktów:
1. Sprawdzenie czy domena domyślna działa (jeśli tak, to należy ją wyłączyć).
2. Usunąć domenę domyślną.
3. Stworzyć nową domenę opierając się o następujące opcje:
   nazwa: domainA
   port baza: 9000
   lokalizacja domeny: as-install-parent/mojedomeny
   (Podczas tworzenia kolejnych domen nie potrzeba wpisywać loginu i hasła, będą one przetrzymywane w pliku .asadminpass)
4. Dokonać deployingu oraz undeployingu aplikacji znajdującej się w źródłach trzema różnymi sposobami, proszę na przemian wykonywać deployingu i undeployingu aplikacji.
5. W pliku logów wskazać kolejno wszystkie czynności wykonywane w tym zadaniu.

Do wykonania podpunktu 4. korzystając z narzędzia asadmin należy dodać przełącznik --port wskazujący odpowiedni dla domeny port administracyjny (9048) oraz wykorzystać przełącznik --domaindir do wskazania lokalizacji domeny.
1.5.3. Zadanie 3. Testowanie wydajności serwera

Zadanie polega na napisaniu skryptu powłoki basha, który ma za zadanie dokonać w pętli deployingu przykładowej aplikacji. Ilość przejść pętli ma być podawana z linii komend. Zadanie ma na celu sprawdzenie, przy jakiej liczbie aplikacji zostanie utracona stabilność pracy serwera albo serwer nie będzie w stanie przyjmować nowych komponentów. Proszę wskazać główne źródła ograniczeń z których wynika utrata stabilności albo odmowa deployingu.

1.5.4. Zadanie 4. Monitorowanie serwera


Należy odpowiedzieć na pytania:
- Jak będzie się zachowywał serwer po instalacji dużej liczby aplikacji?
- Jaki jest wpływ przydzielonej pamięci na jego możliwości?
- Co oprócz pamięci będzie miało wpływ na jego możliwości?
2. NetBeans

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące korzystania z jednego z popularniejszych środowisk IDE, jakim jest NetBeans. Popularność swoją zawdzięcza bardzo rozbudowanemu narzędziu CodeAssistance oraz możliwości integracji z wieloma narzędziami zewnętrznymi. Instrukcja zawiera wiadomości o podstawowych narzędziach tego środowiska wraz z przykładami. Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej części laboratorium.

2.1. Tematyka laboratorium

Netbeans jest projektem rozwijanym w idei wolnego oprogramowania. Jego główne składniki to:

- NetBeans IDE – zintegrowane środowisko programistyczne (IDE). Jeste napisane w Javie, jednak umożliwia ono również tworzenie programów w innych językach.

Netbeans jest rozpowszechniany w dwóch licencjach CDDL lub GPL. Projekt Netbeans był rozwijany w ramach firmy Sun Microsystems, obecnie Oracle.

W ramach projektu NetBeans rozwijane są również rozszerzenia dla IDE. Główne rozszerzenia to:

- NetBeans Profiler – profiler pozwalający na zarządzanie wątkami, procesorem, oraz pamięcią.
- NetBeans Enterprise Pack – rozszerzenie, które wspiera technologie Javy EE. Pozwala na tworzenie aplikacji w architekturze SOA, użycia XML BPEL i usług webowych, tworzenia schematów UML.

2.2. Cel laboratorium

Celem laboratorium jest poznanie w praktyce omówionych narzędzi i ich użycia na przykładach. Narzędzia te są na co dzień używane w pracy programistów, więc ich poznanie jest korzystne z punktu widzenia przyszłej pracy programisty. Podczas laboratorium zapoznasz się z:

- tworzeniem różnych aplikacji w IDE,
- użyciem narzędzia profilera,
- integracją IDE z narzędziami zewnętrznymi.

Dzięki laboratorium nabędziesz praktycznych umiejętności w optymalizowaniu aplikacji przy pomocy profilera.

2.3. Zagadnienia do przygotowania

Przed przystąpieniem do realizacji laboratorium należy zapoznać się z zagadnieniami dotyczącymi:

- SVN,
- GlassFish,
- Ant.

2.4. Opis laboratorium

Laboratorium przedstawia podstawy niezbędne w użytkowaniu IDE – NetBeans w wersji 7.0.1. Zапрęzentowane są niezbędne narzędzia w pracy programisty ułatwiające pisanie kodu, badanie wydajności napisanego kodu. Ostatnia część wstępu teoretycznego przedstawia proces integracji IDE z serwerem aplikacyjnym GlassFish, narzędziem Ant oraz SVN. W celu wykonania ćwiczeń praktycznych przydatne mogą być przedstawione w tym podrozdziale informacje teoretyczne.


2.4.1. Instalacja NetBeans

Na stronie http://netbeans.org/downloads/ jest dostępnych kilka wersji IDE:

- JavaSE – wersja wspiera wszystkie standardowe funkcje Javy SE.
- JavaFX – oprócz wsparcia JavaSE zawiera JavaFX SDK.
- Java – zapewnia narzędzia do zarządzania projektami w technologii Java SE, Java EE, Java ME. Zawiera serwer GlassFish oraz Tomcat.
- Ruby – wersja dla języka Ruby.
NetBeans

- C/C++ – wsparcie dla C/C++, Fortran, asembler, biblioteki Qt.
- PHP – wsparcie dla języka PHP wraz z frameworkami Zend, Symfony.
- All – wersja która zawiera wsparcie dla wszystkich technologii.

1. Ze strony http://netbeans.org/downloads pobieramy wersję Java EE.
2. Plikowi nadajemy prawa +x.
   `chmod +x <nazwa pliku>`
3. Uruchamiamy plik
   `$ ./netbeans-7.0.1-ml-javaee-linux.sh`
5. W kolejnych oknach przechodzimy przez licencje, konfiguracje ścieżek IDE i położenia JDK, przez instalację GlassFish, itp.

Jeśli pobraliśmy środowisko NetBeans w pliku ZIP, to instalacja nie jest wymagana.

2.4.2. Tworzenie nowych projektów aplikacji

W trakcie zakładania nowego projektu mamy kilka typów szablonów projektów do wyboru.
Opcja „Java → Java Application” to aplikacja w technologii Java SE, w standardowym IDE. Poza platformą Netbeans oraz JDK nie wymaga nic więcej.
Opcja „Java → Java Desktop Application” tworzy szkielet aplikacji bazujący na frameworku Swing. Automatycznie tworzona jest podstawa do generowania GUI aplikacji, tworzony jest zestaw klas, zasobów oraz ustawień. Okno po stworzeniu projektu różni się od innych okien, gdyż zawiera dodatkowy panel umożliwiający graficzne tworzenie GUI.
Opcja „Java → Java Class Library” jest podobny do opcji „Java → Java Application”, ale nie posiada opcji do tworzenia main class. Projekt służy do tworzenia bibliotek Java w technologii Java SE.

Projekty Java ME wymagają wcześniejszego aktywowania Java ME oraz zainstalowania J2ME. W tej instrukcji aplikacje mobilne zostały pominięte.
2.4.3. Importowanie bibliotek

Importowanie bibliotek można wykonać na dwa sposoby: dla jednego konkretnego projektu albo o zasięgu globalnym.


Aby uzyskać zarządzanie globalne bibliotekami, należy przejść do Tools → Libraries. W tym menu możemy swobodnie tworzyć usuwać, edytować biblioteki, a w ramach pojedynczych bibliotek edytować takie obiekty jak Classpath, Sources, Javadoc.

2.4.4. Profiler

NetBeans IDE zawiera w sobie potężne narzędzie do profiliingu aplikacji. Dostarcza ono wiele ważnych informacji o aplikacji w czasie wykonania.

W celu skorzystania z profilera potrzeba projektu, który zostanie poddany profilingowi. Przy pierwszym użyciu profilera, IDE musi wykonać operacje inicjalizujące.

Aby uzyskać poprawne wyniki potrzebne są dane kalibracji dla każdej platformy Javy, która będzie używana. W tym celu należy z menu głównego uruchomić Profile → Advenced Commands → Run Profiler Calibration. Pojawi się okno kalibracji, w którym należy wybrać wersje Javy. Po kalibracji zostaniemy poinformowani o jej wyniku (błędy, dane kalibracji). Dane kalibracji zostaną zapisane w pliku .nbprofile w katalogu domowym.

Wykonanie procesu kalibracji wymagane jest tylko przy pierwszym użyciu, jednakże jeżeli zostaną wprowadzone zmiany w maszynie, które mogą wpłynąć na platformę Javy powinno się uruchomić proces kalibracji ponownie.

Wykonując profilowanie projektu pierwszy raz zobaczymy okno informujące nas o tym, że IDE potrzebuje zintegrować projekt z narzędziem do profiliingu. IDE podczas integracji modyfikuje build script projektu i dodaje do niego wpis importujący biblioteki wymagane przez narzędzie do profiliingu. Przed modyfikacją tworzy backup oryginalnego skryptu. Można odwrócić tą operację z poziomu IDE poprzez uruchomienie opcji z głównego menu Profile → Advanced Commands → Unintegrate.

IDE zapewnia bardzo wiele opcji profiliingu, aby dostosować to narzędzie do własnych potrzeb. Jednakże poprawne i świadomie zmieniając tych opcji wymaga dużej wiedzy. Dla większości aplikacji ustawienia domyślne są wystarczające.
Kiedy użyjemy funkcji dostępnej pod Profile → Profile Main Project …, pojawi się główne okno narzędzi profilingu z trzema głównymi zadaniami. Poniższa tabela opisuje te zadania:

<table>
<thead>
<tr>
<th>Zadanie</th>
<th>Rezultat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor Application</td>
<td>Zadanie do uzyskania szczegółowych informacji o właściwościach JVM, aktywności wątków i alokacji pamięci.</td>
</tr>
<tr>
<td>Analyze CPU Performance</td>
<td>Zadanie do uzyskania szczegółowych informacji o wydajności aplikacji, czasie wykonania metod i wiele pomiarów czasowych wywołania metody.</td>
</tr>
<tr>
<td>Analyze Memory Usage</td>
<td>Zadanie do uzyskania szczegółowych informacji o alokacji pamięci oraz mechanizmie garbage collection.</td>
</tr>
</tbody>
</table>

Kiedy uruchomimy Run, zadanie monitoringu wykona się bez podawania kolejnych instrukcji. Ponieważ zadanie monitoringu nakłada bardzo małe narzuty czasowe i pamięciowe, można uruchomić w tym trybie aplikację i przez długi czas korzystać z niej. Kiedy uruchomimy profilera to stworzy on zakładkę w panelu po lewej stronie w IDE. Ta zakładka zawiera opcje pozwalające na kontrolę zadania profilingu, odczyt statusu bieżącego zadania, pokazanie rezultatów zadania, zarządzanie rezultatami profilingu w snapshotach, wyświetlenie podstawowych statystyk. Używając opcji w tym panelu można wyświetlić wykres danych generowany w czasie rzeczywistym. Na jego podstawie można monitorować pamięć, wątki, klasy. Najeżdżając na wykres kursorem możemy odczytać bardziej szczegółowe dane z wykresu w danym punkcie.

Po wybraniu analizy dla CPU dostępne są opcje, które elementy chcemy analizować w czasie rzeczywistym. Możemy analizować całą aplikację, jej część albo konkretny kawałek kodu. Aby zanalizować CPU performance, trzeba wybrać jaki obszar aplikacji chcemy przeanalizować. Dostępne są dwie opcje:

- **Entire application** – w tej opcji wszystkie metody są analizowane. IDE powiadamia o tym, kiedy wątki wchodzą i wychodzą z metod aplikacji pozwalając tym samym na analizę czasową.

- **Part of application** – w tej opcji można dokonać wyboru, która część kodu zostanie przeanalizowana. Ta opcja może znacznie obniżyć narzut wynikający z profilingu. Aby dokonać profilingu części aplikacji trzeba najpierw podać jedną lub więcej tak zwanych „root methods”. Root method jest to metoda, klasa albo package w kodzie źródłowym aplikacji który zostaje określony jako root. Dane profilera są gromadzone kiedy wątki wchodzą i wychodzą z części kodu która została określona jako root.

Przykładowo wykonanie profilowania części klas, ograniczając profilowanie tylko do metod odbędzie się wg poniższego schematu:
1. Zatrzymanie wszystkich sesji profilowania, jeśli jakieś są uruchomione.
2. Przejście do Profile → Profile Main Project.
3. Wybranie profilingu CPU.
4. Wybranie analizy części aplikacji i definicja metod roota, poprzez wybranie opcji define.
5. Kolejno zaznaczenie metod z projektu, Add From Project …, z drzewa zaznaczenie metod które nas interesują i rozpoczęcie analizy.
W opcjach analizy istnieje pole Filter, w którym dokonujemy filtracji danych wyjściowych profilera. Możemy skorzystać z gotowych filtrów, edytować je albo stworzyć własne.

Po rozpoczęciu analizy IDE ładuje aplikacje i uruchamia narzędzie do profilingu. Aby zobaczyć wyniki tworzone w czasie rzeczywistym należy kliknąć na kartę Live Results. Domyślnie wyniki są wyświetlone co kilka sekund. Podczas analizy CPU, okno rezultatów pokazuje informacje o czasie wykonania każdej metody oraz o ilości wywołań danej metody.

Analiza użycia pamięci dostarcza informacji o ilości pamięci zaalokowanej, typach i lokalizacji stworzonych obiektów. Przy analizie pamięci dostępne są dwa tryby analizy. Różnią się od siebie tylko narzutem narzędzia do profilingu oraz ilością otrzymanych danych po analizie. Opcje te są następujące:

- **Record object creation only** – przy tej opcji wszystkie klasy już załadowane oraz każda nowa klasa, która dopiero zostanie załadowana dostarczają informacje o alokacji obiektów.
- **Record both object creation and garbage collection** – przy tej opcji analiza dostarczy dane, o żywotności obiektów, takie jak np.: ile obiektów każdego z typów zostało stworzone, które z nich nadal są „żywe”, ich rozmiar i średni wiek, a także informacje o zaalokowanej pamięci.

Pierwsza opcja funkcjonalnie jest podzborem drugiej, dodatkowo pierwsza opcja posiada mniejszy narzut wydajnościowy i pamięciowy. Kontrolka w postaci paska w głównym oknie Profiling tasks pokazuje aproksymację narzutu spowodowanego narzędziem profilingu.

Przykładowe wykonanie profilingu użycia pamięci z opcją analizy tworzenia i usuwania obiektów może być wykonana według przykładowego schematu:

1. Zatrzymanie, wcześniejszej sesji (jeśli jest uruchomiona).
2. Przejście do narzędzia profilingu poprzez kliknięcie w głównym menu Profile → Profile Main Project.
3. Wybranie analizy pamięci i opcji analizy obu zadań tworzenia i usuwania obiektów.
4. Pozostawić opcje Track every N object … ustawioną na 10. Ta opcja ustawia interwał pobierania próbek na stos. Ustawienie tej opcji na 10 oznacza, że dla każdej klasy tylko co 10 alokowany obiekt tej klasy będzie analizowany dokładnie. Aby analizować każdy alokowany obiekt należy ustawić tę opcję na 1 jednak to spowoduje znaczny wzrost narzutów pochodzących od narzędzia do profilingu.
5. Opcja Record stack trace for allocations służy do zbierania informacji o metodach które wywołują zaalokowane obiekty.
6. Uruchamiamy analizę przez kliknięcie Run.

Kolejno IDE tworzy nową sesję narzędzia do profilingu i przekazuje do niej sterowanie. Aby zobaczyć informacje uzyskane z analizy w czasie rzeczywistym wystarczy kliknąć w oknie profilera (karty po lewej stronie obszaru roboczego), przycisk Live Results. W oknie Live Results możemy odczytać informację o rozmiarze i ilości obiektów, które analizowany projekt alokuje.
Kiedy sesja profilera jest uruchomiona, a analiza jest w trakcie wykonywania, można pobrać wyniki poprzez migawki (snapshots). Dane uchwycone w migawce obrazują dane w chwili wykonania migawki, nie będą one już zmieniane później. Migawki różnią się od wyników generowanych w czasie rzeczywistym w sposób:

- Migawki mogą być wykonane, kiedy sesja nie jest uruchomiona.
- Migawki zawierają dokładniejsze dane
- Migawki mogą być łatwo porównywane

Migawki można zapisywać w postaci plików *.nds i w dowolnej chwili odczytywać.

2.4.5. SVN w NetBeans IDE

NetBeans IDE zapewnia integrację z klientami Subversion w wersji 1.5.x lub późniejszej. Wsparcie Subversion z poziomu IDE pomaga zwiększyć efektywność tworzenia oprogramowania dla zespołów, które pracują ze współdzielonym repozytorium. Pozwala to wykonywać zadania związane z kontrolą wersji przy wykorzystaniu IDE.

Przed rozpoczęciem pracy z SVN należy upewnić się, że na maszynie zainstalowany jest klient SVN. Wsparcie SVN w IDE polega na komunikacji z klientem SVN do którego przekazywane są zadania. Należy się także upewnić, że skonfigurowano repozytorium Subversion, do którego możemy się podłączyć.

NetBeans automatycznie próbuje znaleźć lokalizację pliku wykonawczego Subversion używając zmiennej systemowej PATH. Pomimo tego może zaistnieć konieczność ręcznego określenia ścieżki dostępu.

Domyślnie pliki te znajdują się w:

```
/usr/bin/ lub /usr/local/bin/
```

Aby określić lokalizację z Subversion w IDE należy:

1. Wybrać Tools → Options z głównego menu.
3. W polu Specify the SVN Home Folder wpisujemy lub wybieramy folder z lokalizacją SVN.
4. Restartujemy IDE, aby zmiany odniosły efekt.

Przed rozpoczęciem pracy z wykorzystaniem Subversion należy zapewnić sobie dostęp do repozytorium. IDE nie pozwala na tworzenie repozytoriów Subversion. Do stworzenia lokalnego repozytorium można wykorzystać narzędzie svnadmin, które jest zawarte w kliencie SVN.

Polecenie do tworzenia repozytorium ma poniższą składnię:

```
svnadmin create /path/to/your/repository
```

Praca z systemem kontroli wersji polega na:

- synchronizacji lokalnych plików z repozytorium (checkout),
- dokonywaniu zmian w lokalnych plikach,
- wysyłaniu zmienionych plików do repozytorium (commiting).
Jeśli korzystamy już z SVN w jakimś projekcie, jednak nie jest on rozwijany w IDE, możemy go otworzyć w NetBeans. Dzięki temu wszystkie funkcje związane z SVN będą dostępne. IDE przeskanuje otwarte projekty w poszukiwaniu folderów .swn oraz statusów plików, a wsparcie dla SVN zostanie automatycznie włączone dla projektu.

Aby połączyć się ze zdalnym repozytorium, sprawdzić pliki oraz rozpocząć prace z nimi należy:

1. W NetBeansie wybrać w menu głównym Team → Subversion → Checkout. Otwórz się kreator.
2. W pierwszym panelu kreatora należy podać URL, który zawiera protokół oraz lokalizację repozytorium, do którego chcemy się połączyć.

IDE wspiera następujące protokoły dostępu do repozytorium svn:

<table>
<thead>
<tr>
<th>Protokół</th>
<th>Metoda dostępu</th>
<th>Przykład</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>bezpośredni dostęp do repozytorium (na lokalnym dysku)</td>
<td>file:///repository_path</td>
</tr>
<tr>
<td>http</td>
<td>dostęp przez protokół WebDAV do serwera Subversion</td>
<td><a href="http://hostname/repository_path">http://hostname/repository_path</a></td>
</tr>
<tr>
<td>https</td>
<td>dostęp przez HTTP z szyfrowaniem SSL</td>
<td><a href="https://hostname/repository_path">https://hostname/repository_path</a></td>
</tr>
<tr>
<td>svn</td>
<td>dostęp poprzez własny protokół do serwera svnserve</td>
<td>svn://hostname/repository_path</td>
</tr>
<tr>
<td>svn+ssh</td>
<td>dostęp z użyciem protokołu SVN przez zewnętrzny tunel SSH</td>
<td>svn+ssh://hostname/repository_path</td>
</tr>
</tbody>
</table>

W zależności od wykorzystywanego protokołu może być konieczne podanie dodatkowych informacji, takich nazwa użytkownika i hasło (http://, https:// lub svn://) lub w przypadku svn+ssh:// konieczne jest wcześniejsze ustanowienie zewnętrznego tunelu.

3. W przypadku używania Proxy należy kliknąć Proxy Configuration, a następnie podać niezbędne informacje.
5. W tym samym oknie należy podać numer wersji (Revision). Jeżeli to pole pozostanie puste operacja będzie dotyczyć najnowszej wersji.
6. W polu Local Folder należy podać lokalizację, w której będzie przetrzymywana wersja lokalna.

Istnieje także możliwość importowania projektu do zewnętrznego repozytorium, a następnie kontynuacji pracy nad projektem po synchronizacji.

Uwaga: Podczas gdy tak naprawdę pliki są eksportowane z systemu, słowo „import” w systemie kontroli wersji jest wykorzystywane, aby podkreślić, że pliki są importowane do repozytorium.
Jeśli wersjonowany plik zostanie otworzony w Edytorze IDE, istnieje możliwość oglądania na bieżąco zmian wprowadzanych w pliku w stosunku do ostatnio sprawdzonej wersji z repozytorium.


Po wprowadzeniu zmian do kodu źródłowego należy go przenieść do repozytorium. Dobrą praktyką jest uaktualnienie wszystkich kopii przed dokonaniem commit, aby upewnić się, że nie powstaną żadne konflikty. Konflikty mogą pojawić się szczególnie jeśli nad projektem pracuje jednocześnie kilku programistów. IDE zawiera także Conflict Resolver, który pozwala na bezpieczne rozwiązanie wszystkich napotkanych konfliktów.

Uaktualnienia można dokonać poprzez wybranie Subversion → Update z menu prawego przycisku mysz każdego wersjonowanego projektu. Aby dokonać aktualizacji można także kliknąć ikonę Update All w oknie Versioning. Wszystkie wprowadzone zmiany w repozytorium zostaną wyświetlone w oknie Versioning Output.

Podczas operacji update lub commit, IDE porównuje pliki lokalne z plikami w repozytorium, aby sprawdzić czy nie zasłyły inne zmiany. Po wprowadzeniu zmian w plikach podczas pracy ze starszą wersją, a następnie próbą wykonania commit w przypadku gdy najnowsza wersja wprowadza zmiany w tych samych linijkach, próba ta zakończy się konfliktem.


Rozwiązywanie konfliktów polega na akceptowaniu jednej z dwóch wersji wyświetlanych na ekranie. Po rozwiązaniu wszystkich konfliktów można dokonać commit zmodyfikowanych plików.

Po wprowadzeniu zmian w plikach oraz rozwiązaniu ewentualnych konfliktów należy wykonać commit plików do repozytorium. Po wybraniu opcji Commit (np. z menu projektu) otwarte zostanie okno dialogowe wyświetlające pliki, które zostaną uwzględnione.

Okno Commit wyświetla listę:
- wszystkich lokalnie zmienionych plików,
- wszystkich plików, które zostały lokalnie usunięte,
- wszystkie nowo stworzone pliki (które nie istnieją jeszcze w repozytorium),
- wszystkie pliki, których nazwa została zmieniona.
2.4.6. Integracja IDE z GlassFish v3

W celu integracji należy dodać w NetBeans serwer, z którym będziemy pracować. Dokonać tego można na kilka sposobów.

2. W nowo otwartym oknie wybieramy serwer, który będziemy używać (GlassFish Server 3) oraz wpisujemy jego nazwę.
3. Podajemy ścieżkę do katalogu z zainstalowanym GlassFishem, a następnie klikamy Next.
4. W następnym oknie wybieramy domenę, którą będziemy wykorzystywać. Mamy do wyboru dwie opcje:
   b. Register Remote Domain - ta opcja pozwala pracować na domenie znajdującej się na innej maszynie. W polu Host Name należy podać adres serwera. W polu Admin Port podajemy numer portu administracyjnego, który identyfikuje jednoznacznie nasza domenę.

2.5. Przebieg laboratorium

Drugą część instrukcji zawiera zadania do praktycznej realizacji, które demonstrują zastosowanie technik z omawianego zagadnienia.

2.5.1. Zadanie 1. Zarządzanie projektami

Zadanie polega na wykonaniu kilku małych zadań, które przetestują wiedzę teoretyczną nabytą w pierwszej części laboratorium.

2.5.2. Zadanie 2. Użycie Profilera

W tym zadaniu proszę wykonać kolejne podpunkty.

1. Uruchomić dowolną aplikację (może być to jedna z aplikacji przykładowych zdefiniowanych w IDE) i przeprowadzić profiling typu Monitor, CPU i Memory. Zapoznać się z generowanymi wynikami i wytłumaczyć, co który element oznacza.


3. Dokonać analizy pamięciowej i czasowej wykonania kodu:
```java
public class Strings {
    public static void main(String[] args) {
        String s = "SOA";
        for(int i=0; i<10000; i++)
            s = s + "SOA";
    }
}
```

Korzystając z narzędzi profilingu proszę znaleźć nieoptymalne użycie funkcji Javy, a następnie proszę poprawić kod, aby wykonywał się w krótszym czasie i wymagał mniejszych zasobów pamięciowych. Wyniki proszę przedstawić w sprawozdaniu.
3. Servlyety

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości dotyczące programowania aplikacji webowych wykorzystujących technologie Java Servlet. Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej części laboratorium.

3.1. Tematyka laboratorium

Tematyką laboratorium jest programowanie aplikacji webowych opartych na technologii Java Servlet, która zapewnia oparte na komponentach, niezależne od platformy metody na ich budowaniu bez ograniczeń programów CGI. Servlyety posiadają wszystkie zalety Javy, w tym: przenośność, wydajność, możliwość ponownego wykorzystania kodu i ochrony przed awariami. Ponieważ servlyety są niezależne od platformy oraz serwera, daje to możliwość wyboru środowiska.

Zadaniem servlyetów w aplikacjach webowych jest najczęściej:
- Odczytywanie danych wysłanych przez użytkownika.
- Wyszukiwanie i przetwarzanie informacji o żądaniu tj. możliwości przeglądarki, cookies, nazwa hosta.
- Generowanie wyników.
- Formatowanie wyników jako dokument.
- Ustawienie odpowiednich parametrów odpowiedzi http.
- Odesłanie dokumentu do klienta.

3.2. Zagadnienia do przygotowania

Przed przystąpieniem do realizacji laboratorium należy zapoznać się z zagadnieniami dotyczącymi:
- aplikacji NetBeans
- serwera aplikacji GlassFish
- nagłówki HTTP oraz metody GET, POST
- mechanizm Cookies
- protokołu HTTP/1.1
- podstawy języka HTML

Literatura:
[1] IETF, RFC 2616, „Hypertext Transfer Protocol -- HTTP/1.1”
[2] IETF, RFC 2965, „HTTP State Management Mechanism”
[5] Sun, JSR 315, „Java Servlet 3.0 Specyfications”
3.3. Opis laboratorium


Servlety są technologią Javy opartą o komponenty Web, które generują dynamiczną treść i są zarządzane przez kontener. Tak jak pozostałe technologie Java, servlety są niezależne od platformy. Servlety są klaskami Java kompilowanymi do bytecode, który może być dynamicznie załadowany oraz uruchomiony przez serwer. Interakcja pomiędzy servletami a klientem Web odbywa się z wykorzystaniem paradygmatu żądanie/odpowiedź implementowanego przez kontener.

Wiele żądań klientów może być spełniona zwracając gotowy dokument i te żądania mogą być obsłużone przez serwer bez wywoływania servletu. Jednak w wielu przypadkach statyczny wynik nie jest wystarczający i strona musi być wygenerowana dla każdego żądania. Jest wiele powodów, dla których strona musi być generowana w locie:

- strona bazuje na danych wprowadzonych przez użytkownika,
- strona bazuje na danych które zmieniają się często,
- strona wykorzystuje dane z bazy danych lub innej usługi serwera.

Zasadą jest, że servlety nie są ograniczone do serwerów www lub serwerów aplikacji, które obsługują żądania HTTP. Mogą one być również użyte z innymi typami usług np. serwerem poczty lub FTP.

API do obsługi servletów składa się z dwóch pakietów javax.servlet i javax.servlet.http.

3.3.1. Cykl życia Servletu

Kiedy klient wydaje żądanie, wywołując servlet, serwer ładuje i wykonuje odpowiednio klasy Javy. Klasy te generują treść, którą serwer wysyła z powrotem do klienta. W większości przypadków klientem jest przeglądarka internetowa, serwerem jest serwer www, a servlet zwraca dokument HTML. Ze względu na to, że strona ma to duże znaczenie, że servlety nie są ograniczone do serwerów www lub serwerów aplikacji, które obsługują żądania HTTP. Mogą one być również użyte z innymi typami usług np. serwerem poczty lub FTP.

API do obsługi servletów składa się z dwóch pakietów javax.servlet i javax.servlet.http.
zażądania. Dla każdego nadchodzącego połączenia do servletu wywoływana jest metoda service() na tym servlecie. Metoda destroy() jest wywoływana, aby zwolnić zasoby przy zamykaniu serwera.

3.3.2. Podstawowe API Servletów

Metoda init()

Metoda init jest wywoływana, gdy servlet jest tworzony i nie jest wywoływana ponownie dla każdego żądania użytkownika. Servlet może zostać stworzony, gdy użytkownik pierwszy raz go wywoła lub gdy serwer zostanie uruchomiony – zależy to od tego, w jaki sposób servlet został skonfigurowany na serwerze.

Istnieją dwie wersje metody init: pierwsza, która nie przyjmuje żadnego argumentu oraz druga, która przyjmuje argument ServletConfig. Pierwsza wersja jest używana jeśli servlet nie potrzebuje odczytywać żadnych ustawień zależnych od serwera. Druga wersja init jest używana gdy servlet używa ustawień specyficznych dla serwera. Na przykład servlet może potrzebować informacji o ustawieniach bazy danych, plikach hasel, serializowanych ciasteczkach poprzednich żądań.

Parametr ServletConfig posiada metodę getInitParameter, która umożliwia dostęp do parametrów inicjalizacyjnych servletu. Zarówno nazwa jak i wartość tych parametrów są Stringami.

Jeżeli piszesz metodę init która przyjmuje ServletConfig, jako argument należy zawsze wywoływać super.init w pierwszej linijce.

Metoda service()

Za każdym razem gdy serwer dostaje żądanie o servlet, serwer tworzy nowy wątek i wywołuje metodę service(). Metoda ta sprawdza typ żądania HTTP (GET, POST, PUT, DELETE itp.) i wywołuje odpowiednio funkcje: doGet, doPost itd. Jeżeli servlet musi obsługiwać zarówno żądania GET jak i POST, dobrą praktyką jest wywołanie metody doGet() w metodzie doPost() lub odwrotnie. Nie należy natomiast nadpisać metody service(). Takie postępowanie można stosować wtedy gdy obsługa żądań GET i POST przebiega tak samo. Poniżej przedstawiono prawidłowe postępowanie:

```java
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // Kod servletu
}
public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    doGet(request, response);
}
```

Takie postępowanie ma następujące zalety:
1. Istnieje możliwość późniejszego dodania obsługi innych usług poprzez dodanie doPut, doTrace. Nadpisanie service() wyklucza tę możliwość.
2. Można dodać wsparcie dla modyfikowania poprzez metodę getModified().
3. Uzyskujemy wsparcie żądań HEAD. System zwraca nagłówki i kody statusów, które ustawia doGet, ale omija ciało strony.
4. Uzyskujemy wsparcie żądań OPTIONS. Jeśli metoda doGet() istnieje, standardowa metoda service odpowiada na żądanie OPTIONS zwracając nagłówek Allow oznaczający, że GET, HEAD, OPTIONS, TRACE są wspierane.

5. Uzyskujemy wsparcie żądań TRACE. TRACE jest metodą żądania używana do debugowania klientów. Zwraca ona nagłówek żądania HTTP z powrotem do klienta.

Metoda destroy()

Czasami może pojawić się konieczność usunięcia wcześniej załadowanej instancji servletu. Może to być zażądane, administrator może to zrobić lub servlet, gdy był bezczynny przez dłuższy czas. Zanim to się stanie wywoływana jest metoda destroy(). Metoda ta pozwala na zamknięcie otwartego połączenia z bazą danych, zatrzymanie wątków będących w tle, zapisanie listy ciasteczek.

Interfejs SingleThreadModel

Zazwyczaj system tworzy pojedynczą instancję servletu, a potem tworzy nowy wątek dla każdego przychodzącego żądania użytkownika. Jeżeli kilka wątków jest uruchomionych jednocześnie, nowe żądania przychodzą, podczas gdy poprzednie są jeszcze wykonywane. Oznacza to, że metody doGet u doPost muszą być ostrożne przy synchronizacji dostępu do pól lub współdzielonych danych ponieważ kilka wątków może żądać dostępu do danych jednocześnie.

Aby temu zapobiec można zaimplementować w serwisie interfejs SingleThreadModel.

Zaimplementowanie tego interfejsu gwarantuje, że nigdy nie wystąpi sytuacja w której więcej niż jeden wątek żądania będzie próbował uzyskać dostęp do servletu. Jest to osiągnięte poprzez kolejkowanie wszystkich żądań i przekazywanie ich kolejno do instancji serwletu lub poprzez utworzenie puli wielu instancji, z których każda obsługuje jednocześnie jedno żądanie. Oznacza to że nie trzeba się martwić o jednoczesny dostęp do zwyczajnych pól servletu. Trzeba natomiast synchronizować dostęp do pól statycznych oraz danych współdzielonych poza serwlem.

Synchroniczny dostęp do servletów może w znaczny sposób obniżyć wydajność jeżeli servlet jest często wywoływany.

3.3.3. Kompilacja i instalacja servletu

Podczas tworzenia servletu używając NetBeansa należy postępować następująco:

1. Należy stworzyć nowy projekt Aplikacji Webowej.
Zazwyczaj servlet tworzy się przez utworzenie klasy dziedziczącej po klasie HttpServlet. Istnieje też klasa GenericServlet, która została zaprojektowana z przeznaczeniem do komunikacji przy użyciu innych protokołów niż tylko HTTP. Można też zaimplementować interfejs Servlet – to rozwiązanie może zostać wykorzystane jeśli mamy zamiar zastosować servlyty w gotowych już rozwiązaniach.

Po stworzeniu nowego servletu i dodaniu go do projektu należy dodatkowo zamieścić informacje w deskryptorze rozmieszczenia (deployment descriptor).

Poniżej przedstawione są informacje, które mogą być zawarte w deskryptorze umiejscowienia:
- parametry inicjalizacyjne ServletContext,
- konfiguracja Sesji,
- deklaracje servletów,
- mapowania servletów,
- klasy Listener,
- definicje filtrów i ich mapowania,
- odwzorowania typów MIME,
- lista plików Welcome,
- strony błędów,
- mapowania lokalne i kodowania,
- konfiguracja bezpieczeństwa.

W tym celu można wykorzystać również mechanizm adnotacji, który został udostępniony w API servletów od wersji 2.5. Używanie adnotacji powoduje, że używanie pliku web.xml staje się opcjonalne.

W aplikacji webowej adnotacje klas zostaną przetworzone tylko wtedy, jeśli klasy te będą umieszczone w folderze WEB-INF/classes lub jeśli zostaną spakowane w plik jar umieszczony w WEB-INF/lib.

Deskryptor rozmieszczenia aplikacji webowej zawiera atrybut „metadata-complete” elementu web-app. Atrybut ten definiuje czy deskryptor jest kompletny, czy pliki class powinny być sprawdzane na wystąpienie adnotacji.

Jeśli atrybut „metadata-complete” ma wartość „true”, wszystkie adnotacje zawarte w plikach class zostaną zignorowane. W przeciwnym wypadku adnotacje zostaną uwzględnione.

Do najbardziej użytecznych należą:
- @WebFilter – deklaruje filtr servletu,
- @WebInitParam – określa parametry inicjalizacyjne servletu lub filtru,
- @WebServlet – deklaruje servlet,
- @WebListener – deklaruje WebListener,
- @MultipartConfig – żądania typu MIME/MULTIPART.

3.3.4. Servlet HTTP

HttpServlet rozszerza klasę GenericServlet – zawiera ona metody do obsługi danych HTTP. Metody doGet(), doPost(), doPut() itd. obsługują poszczególne typy żądań HTTP. Te metody są wywoływane przez domyślną implementację metody service(), w której rozstrzygane jest jakiego typu jest przychodzące żądanie, a następnie wywołuje odpowiednią metodę. Poniżej przedstawiony jest prosty servlet.
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class HelloWorldServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        resp.setContentType("text/html");
        PrintWriter out = resp.getWriter();
        out.println("<html>");
        out.println("<head><title>Pozdrowienie</title></head>");
        out.println("<body><h1>Hello World!</h1><h6>Ponownie</h6></body></html>" );
    }
}

Metoda doGet() jest wywoływana za każdym razem gdy URL wskazuje na ten servlet. HttpServletRequest i HttpServletResponse zawierają podstawowe informacje o żądaniu oraz zapewniają możliwość odpowiedzi.

Metoda doGet() w tym przykładzie odpowiada za zrobienie trzech rzeczy.
- Ustawia typ odpowiedzi na text/html.
- Wywołuje metodę getWriter() aby uzyskać obiekt PrintWriter
- Używa strumienia aby wysłać odpowiedź HTML do klienta

Zaimplementowanie metody getLastModified() pozwala poinformować przeglądarkę, kiedy ostatnio została zmodyfikowana zawartość servletu. Pozwala to na wykorzystanie mechanizmu cache'owania. Wartość zwracana przez tą metodę określa ile minęło milisekund od ostatniej modyfikacji. Domyślną wartością jest -1 (servlet nie posiada informacji kiedy ostatnio był modyfikowany).

Metoda getServletInfo() zwraca informacje o servlecie, autorze i wersji.

3.3.5. Aplikacje webowe

Aplikacja webowa składa się ze zbioru zasobów (w tym servletów), statycznej zawartości, plików JSP i bibliotek klas zainstalowanych pod określoną ścieżką na serwerze. Ta ścieżka nazywana jest kontekstem servletu. Wszystkie servlety zainstalowane z użyciem tego kontekstu są odizolowane od innego oprogramowania znajdującego się na tym serwerze.

Drzewo katalogowe kontekstu servletu zawiera kilka typów zasobów wliczając w to pliki klas, pliki jar, pliki jsp, pliki statyczne (HTML, JPEG).
Najprostsza instalacja servletu tworzy jedynie pojedynczy kontekst w /, co jest szczytem drzewa ścieżek serwera www. Servlyte i statyczna zawartość jest instalowana w tym kontekście.

W systemie plików aplikacja webowa składa się z katalogu, który posiada podkatalog WEB-INF oraz możliwie inną zawartość. Katalog WEB-INF zawiera katalog classes (zawiera kod aplikacji), katalog lib (zawiera pliki jar aplikacji) oraz plik web.xml. Plik ten zawiera wszystkie informacje konfiguracyjne dla tego servletu w tym nazwy, mapowanie ścieżek, parametry inicjalizacyjne oraz informacje konfiguracyjne poziomów kontekstu.

Aplikacje webowe mogą być spakowane w pliki war równoważne plikom jar. Do tego celu można się posłużyć narzędziem jar dostarczonym z JDK lub użyć np. skryptu Ant.

3.3.6. Kontekst Servletu


Asterisk (*) służy jako znak wildcard. URI pasujące do wzorca są mapowane do określonego servletu pod warunkiem, że inne odwzorowanie nie zostało już dopasowane. Gdy kontener serleytow otrzyma żądanie zawsze mapuje je do właściwego servletu zgodnie z następującym porządkiem:

1. dokładne dopasowanie ścieżek,
2. odwzorowanie prefixów,
3. rozszerzenia.

Jeśli nie zostanie znaleziony właściwy servlet, serwer zwróci bląd. Jeśli servlet będzie odwzorowany na ścieżkę /, wtedy zostanie on domyślnym servelem aplikacji i zostanie wywołany, jeśli żaden inny servlet nie zostanie znaleziony.

Istnieje tylko jedna instancja obiektu interfejsu ServletContext powiązana z aplikacją webową usadowioną na w kontenerze. W przypadku gdy kontener jest rozproszony na kilku maszynach wirtualnych, aplikacja webowa będzie posiadała po jednej instancji ServletContext na JVM. Servlyte w kontenerze, który nie został usadowiony jako część aplikacji webowej, są częścią domyślnej aplikacji webowej i posiadają domyślny ServletContext. W rozproszonych kontenerach, domyślny ServletContext nie jest rozproszony i musi istnieć na jednej JVM.

Zasoby w kontekście serleytow (tj. pliki HTML, zdjęcia oraz inne dane) mogą być dostępne bezpośrednio poprzez serwer. Jeżeli plik index.html jest przechowywany w katalogu /app kontekstu, dostęp do niego można uzyskać poprzez /app/index.html. Zasoby te mogą być też dostępne przy pomocy obiektu ServletContext, za pomocą metod getReasource() oraz getResourceAsStream().
Interfejs ServletContext zapewnia servletom dostęp do informacji o lokalnym środowisku. Interfejs ten posiada także szereg metod pozwala 


gambaranym atrubutów. Metody getInitParameter() oraz getInitParameterNames() pozwalają servletom na uzyskanie parametrów inicjalizacyjnych.

Interfejs ServletContext pozwala na programowe dodawanie do kontekstu oraz konfigurowanie takich elementów jak Servlety, Filtry, Listeners.

Servlet może przypisać obiekt atrubutu do kontekstu używając nazwy. Każdy atrubut przypisany do servletu jest dostępny dla wszystkich servletów, które są częścią tej samej aplikacji Webowej.

Poniższe metody pozwalają na dostęp do tych danych:

- getAttribute,
- getAttributeNames,
- setAttribute,
- removeAttribute.

Atrybuty kontekstowe są lokalne dla JVM, w której są stworzone. Zapobiega to sytuacji, w której atrybuty ServletContext będą dzielone w rozproszonym kontenerze. Jeśli informacje muszą być współdzielone w rozproszonym środowisku, powinny być zapisane z użyciem sesji lub przechowane w bazie danych.

Istnieje możliwość uporządkowanej listy częściowych URI nazywanych plikami welcome w deskryptorze rozmieszczenia aplikacji.

Celem tego mechanizmu jest umożliwienie określenia uporządkowanej listy częściowych URI dla kontenera, dodawanych do URI, gdy żądanie nie jest odwzorowane na komponent.

Plik welcome „index.html” został zdefiniowany tak, że żądanie o URL host:port/webapp/directory/ (gdzie 'directory' jest wpisem w WAR, który nie jest mapowany na servlet ani stronę JSP) jest zwrócone do klienta jako 'host:port/webapp/directory/index.html'.

Jeśli kontener Web otrzyma prawidłowe, częściowe żądanie, kontener sprawdzi listę plików welcome w deskryptorze rozmieszczenia. Lista ta zawiera częściowe URL bez poprzedzających bądź następujących znaków '/'. Serwer musi dodać każdy plik welcome w porządku określonym w deskryptorze do częściowego żądania i sprawdzić, czy statyczny zasób w archiwum WAR jest odwzorowany na ten URI. Jeśli nie zostało znalezione dopasowanie, serwer ponownie doda każdy z plików welcome i sprawdzi odwzorowania. Kontener prześle żądanie do pierwszego dopasowanego zasobu.

Poniżej przedstawiono przykład zdefiniowanych plików welcome w deskryptorze rozmieszczenia:

```xml
<welcome-file-list>
  <welcome-file>index.html</welcome-file>
  <welcome-file>default.jsp</welcome-file>
</welcome-file-list>
```

WAR zawiera następującą zawartość statyczną:

```
/foo/index.html
/foo/default.jsp
/foo/orderform.html
```
3.3.7. Żądanie servletu

Aby servlet mógł wygenerować właściwą odpowiedź, wymaga często określonych informacji o żądaniu. Może istnieć także potrzeba, aby użytkownik żądający dostępu do servletu został uwierzytelniony.

Interfejsy ServletRequest oraz HttpServletRequest zapewniają dostęp do informacji o żądaniu, które jest obsłudowane. Podczas przekazywania żądania obiekt implementujący jeden z tych interfejsów jest przekazywany przez kontener do servletu. Obiekt ten pozwala na uzyskanie informacji o protokole, URL, typie żądania, poszczególnych fragmentach żądania (nagłówków, strumienia wejściowego) oraz pozwala na dostęp do parametrów żądania.

- getProtocol() - zwraca protokół wykorzystany przez żądanie,
- getRemoteHost() - zwraca nazwę hosta klienta,
- getServerName() oraz getServerPort() - zwraca informacje o serwerze,
- getParameter() oraz getParameterValues() - zwraca parametry żądania.

3.3.8. Odpowiedź servletu

Interfejsy ServletResponse oraz HttpServletResponse dostarczają metod potrzebnych do stworzenia i manipulowania wyjściem servletu. Metodę setContentType() wykorzystuje się do określenia typu danych zwracanych przez servlet. Interfejsy te dostarczają dwóch metod: getOutputStream() oraz getWriter(). Pierwsza z nich zwraca obiekt ServletOutputStream, który może być używany zarówno z danymi tekstowymi jaki i binarnymi. Druga metoda zwraca obiekt java.io.PrintWriter, który może być wykorzystywany wyłącznie z danymi tekstowymi. W przypadku użycia metody getWriter() najpierw powinna być wywołana metoda setContentType(). Interfejs HttpServleResponse zawiera także dodatkowo metody pozwalające na manipulacje nagłówkami HTTP. Metody setHeader(), setIntHeader() oraz setDateHeader() pozwalają nadać wartości określonym nagłówkom, podczas gdy metoda containsHeader() pozwala stwierdzić, czy określony nagłówek ma już ustawnioną wartość.

Jeżeli serwer odpowiada na żądanie od klienta odpowiedź składa się z linijki statusu, nagłówka oraz dokumentu.
HTTP/1.1 200 OK
Content-Type: text/plain
Hello World

Linijka statusu zawiera wersję HTTP oraz kod statusu wraz z krótką wiadomością z nią związaną (OK).

Servlety mogą wykonać wiele zadań manipulując linią statusów oraz nagłówkami odpowiedzi. Umożliwiają m.in. przeniesienie użytkownika na inną stronę, czy wskazanie typu załączonego dokumentu.

3.3.9. Request Dispatching

Budując aplikację webową, użyteczna staje się możliwość przekazania żądania do innego servletu lub przekazanie wyjścia do innego servletu w odpowiedzi. Mechanizm ten jest zapewniony poprzez interfejs RequestDispatcher.

Obiekt implementujący RequestDispatcher może zostać uzyskany z obiektu ServletContext używając następujących metod:
• getRequestDispatcher(),
• getNamedDispatcher().

Metoda getRequestDispatcher() pobiera String jako argument określający ścieżkę w zasięgu ServletContext. Ścieżka ta musi być zależna od korzenia kontekstu servletu i rozpoczynać się od '/' lub być pusta.

Metoda getNamedDispatcher() pobiera String jako argument wskazujący nazwę servletu zawartego w ServletContext.

Metoda forward interfejsu RequestDispatcher może zostać wywołana przez wywołujący ją servlet tylko wtedy, jeżeli nie zostały zatwierdzone do wysłania żadne dane do klienta. Jeśli jakiekolwiek dane wyjściowe, które nie zostały jeszcze zatwierdzone, są w buforze odpowiedzi, to muszą one zostać wyczyszczone przed wywołaniem metody forward. Jeśli dane wyjściowe zostały zatwierdzone, zostanie rzucony wyjątek IllegalStateException.

Elementy ścieżki obiektu żądania przekazane do docelowego servletu muszą odzwierciedlać ścieżkę użytą do uzyskania obiektu RequestDispatcher.

Zanim metoda forward interfejsu RequestDispatcher zwróci wartość, odpowiedź musi zostać wysłana i zatwierdzona.

3.3.10. Określanie kodów statusu

Ponieważ wiadomość jest bezpośrednio związana z kodem statusu, a wersja HTTP jest określona przez serwer, wszystkie servlety muszą ustawiać kod statusu. Aby to zrobić należy wywołać funkcję setStatus na obiekcie HttpServletResponse. Należy zadbać, aby ustawiać kod statusu zawsze przed wysłaniem jakichkolwiek danych poprzez PrintWriter.

Metoda setStatus przyjmuje int jako argument (kod statusu), jednak zamiast używać numerów, można posłużyć się stałymi zdefiniowanymi w HttpServletResponse. Nazwy wszystkich stałych zostały zaczerpnięte ze standardu HTTP 1.1. Wszystkie znaki zostały zamienione na duże litery, dodano przedrostek
SC(Status Code), a wszystkie spacje zamieniono na podkreślenia. Na przykład odpowiednikiem 404 Not Found w HttpServletResponse jest SC_NOT_FOUND. Istnieją dwie dodatkowe metody do obsługi kodów statusu:

```java
public void sendError(int code, String message)
Metoda ta wysyła kod statusu razem z wiadomością podaną jako parametr.
public void sendRedirect(String url)
Metoda ta powoduje wysłanie kodu statusu 302 razem z nagłówkiem Location z podanym URL nowego dokumentu. Powoduje to przekierowanie użytkownika na stronę pod adresem podanym jako argument.
```

3.3.11. Filtrowanie żądań i odpowiedzi

Filtr jest obiektem, który pozwala na przekształcanie żądań bądź odpowiedzi. Filtry różnią się od innych komponentów webowych tym, że zazwyczaj nie tworzą odpowiedzi samodzielnie. Zamiast tego filtr zapewnia funkcjonalność, która może być dołączona do dowolnego zasobu. Jako konsekwencja tego, filtr nie powinien mieć żadnych zależności z zasobem, dla którego jest filtrem. Głównymi zadaniami wykonywanymi przez filtry są:

- blokowanie par żądanie – odpowiedź przed dalszym przetwarzaniem,
- modyfikowanie nagłówków żądania i danych,
- modyfikowanie nagłówków odpowiedzi i danych,
- interakcja z zewnętrznymi zasobami.

Do zastosowania filtrów wlicza się uwierzytelnianie, logowanie, konwersja grafiki, kompresja danych, szyfrowanie, stream tokenizing i transformacje XML.

Istnieje możliwość takiej konfiguracji zasobu, aby był filtrowany przez łańcuch zera, jednego lub większej liczby filtrów w określonym porządku.  Łańcuch ten jest określany wtedy, gdy aplikacja webowa zawierająca ten komponent jest instalowana w trakcie ładowania komponentu przez kontener.

Zadania związane z obsługą filtrów sprowadzają się do poniższych:

- programowanie filtrów,
- programowanie przetworzonych żądań i odpowiedzi,
- określanie łańcucha filtrowania dla każdego zasobu Web.

API filtrowania jest zdefiniowane przez interfejsy Filter, FilterChain i FilterConfig w pakiecie javax.servlet. Filtry definiuje się poprzez implementacje interfejsu Filter. Najważniejszą metodą tego interfejsu jest doFilter, do której jest przekazywane żądanie, odpowiedź oraz łańcuch filtrów.

Interfejs Filter dostarcza trzech metod:

1. `init(FilterConfig cfg)` – jest wywoływana tylko raz przez kontener, kiedy filtr jest włączany do użycia. Argument FilterConfig przesłany do tej metody zawiera parametry inicjalizacyjne filtru. Parametry te można określić podczas konfiguracji używając narzędzia do asemlacji.
2. `destroy()` – wywoływana przez kontener, gdy filtr jest wyłączany z użycia.
3. `doFilter(ServletRequest req, ServletResponse res, FilterChain chain)` – jest wywoływana przez kontener dla każdego żądania servletu odwzorowanego na ten filtr, przed wywołaniem servletu. FilterChain, który jest przekazany do tej metody może być wykorzystany do następnego filtra w łańcuchu.
Metoda ta może wykonać następujące akcje:

- zbadać nagłówki żądania,
- dostosować obiekt żądania,
- dostosować obiekt odpowiedzi,
- wywołać następny filtr w łańcuchu. Jeżeli filtr jest ostatnim w łańcuchu, docelowy zasób będzie dostępny. W przeciwnym wypadku wywoływana jest metoda doFilter następnego filtra. Istnieje możliwość zablokowania dalszego przetwarzania poprzez nie wywoływanie metody doFilter na kolejnym obiekcie łańcucha filtrów,
- zbadać nagłówki odpowiedzi po tym jak został wywołany następny filtr w łańcuchu,
- wyrzucić wyjątek aby zasygnalizować wystąpienie błędu w przetwarzaniu.

Istnieje wiele sposobów na modyfikacje żądania lub odpowiedzi przez filtr. Może zostać to zrobione poprzez dodanie atrybutu do żądania lub danych do odpowiedzi.


Kontener servletów używa mapowania filtrów do przypisywania ich do zasobów. Filtry są mapowane w przypadku komponentów przez nazwy, a w przypadku zasobów sieciowych poprzez wzorzec URL. Wywoływane są one w kolejności w jakiej występują odwzorowania na liście odwzorowań filtrów.

Może się mapować jeden filtr do jednego lub większej ilości zasobów, oraz można mapować więcej niż jeden filtr do zasobu.

Łańcuch filtrów jest jednym z obiektów przekazywanych do metody doFilter. Ten łańcuch jest formalnie pośrednio przez mapowania filtrów. Kolejność filtrów w łańcuchu jest taka sama, jak porządek odwzorowań filtrów w deskryptorze rozmieszczenia aplikacji webowej.

Przy tworzeniu filtra należy umieścić ścieżkę z klasą filtra w zmiennej classpath projektu, co jest automatycznie robione po umieszczeniu pliku w katalogu WEB-INF/classes lub WEB-INF/lib. Filtr jest wtedy dodawany do pliku konfiguracyjnego WEB-INF/web.xml podobnie jak servlet. Do poprawnej konfiguracji niezbędne są dwa bloki.

**Blok <filter>**

Głównymi elementami tego bloku są:

- filter–name – nazwa, która będzie używana do identyfikacji filtra,
- filter–class – nazwa klasy filtru. Powinna zawierać nazwę pakietu. Będzie wykorzystana przez kontener do załadowania klasy

Poniżej przedstawiony jest przykładowy blok:

```xml
<filter>
  <filter-name>Timer</filter-name>
  <filter-class>filter.TimerFilter</filter-class>
</filter>
```
Blok <filter-mappings>
W tym bloku zdefiniowane są odwzorowania filtrów:
- filter-name – nazwa filtra, która musi odpowiadać nazwie podanej w bloku <filter>.
- url-pattern – określa wzorzec URL lub nazwę servletu dla którego filtrowanie będzie stosowane.

Przykład:
```xml
<filter-mapping>
  <filter-name>Timer</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Ponieważ HTTP jest protokołem bezstanowym, dlatego za każdym razem gdy klient otwiera stronę internetową, otwiera też nowe połączenie do serwera www. Serwer automatycznie nie uzyskuje informacji kontekstowej o kliencie. Dzieje się tak również, jeżeli serwer wspiera utrwalanie (keep-alive) połączeń HTTP i utrzymuje gniazdo otwarte dla kilku żądań klienta, które są wysyłane w małych odcinkach czasu.

3.3.12. Cookies

Aby utworzyć ciasteczkę aplikacja webowa musi dołączyć nagłówek Cookie z określoną wartością do odpowiedzi HTTP. Następnie przeglądarka wysyła podobny nagłówek z tą samą wartością z powrotem do serwera przy kolejnych żądaniach. Aplikacja webowa może wykorzystać wartość ciasteczka do śledzenia określonego użytkownika lub do obsługi śledzenia sesji. Ponieważ ciasteczka używają pojedynczego nagłówka Cookie, istnieje możliwość zawarcia kilku par nazwa/wartość jako wartości ciasteczka.

API servletów zawiera klasę javax.servlet.http.Cookie która umożliwia prace z ciasteczkami. Dodatkowo HttpServletRequest zawiera metodę addCookie(), a HttpServletResponse zapewnia metodę getCookies(). Pozwalają one na odczytywanie oraz zapisywanie ciasteczek do nagłówków HTTP.

Funkcja getCookies() zwraca tablicę, dlatego ciasteczka muszą być przeglądane sekwencyjnie. Poniżej przedstawiony jest przykład:
```java
Cookie[] cookies;
cookies = req.getCookies();
String userid = null;
for (int i = 0; i < cookies.length; i++)
    if (cookies[i].getName().equals("userid"))
        userid = cookies[i].getAttribute();
```

Ciasteczka mogą być odczytywane w każdej chwili, jednak tworzone mogą być tylko jeśli nie zostało jeszcze nic wysłane do klienta. Powodem tego jest to, że ciasteczka są wysyłane z użyciem nagłówków HTTP, a nagłówki te mogą być wysłane tylko przed wysłaniem pozostałej treści odpowiedzi.
Poniżej przedstawiony jest przykład tworzenia ciasteczka:

```java
String userid = createUserID();
Cookie c = new Cookie("userid", userid);
resp.addCookie(c);
```

Należy zwrócić uwagę, że przeglądarka ma ograniczenia co do obsługiwanych ciasteczek. W tym co do ich ilości oraz rozmiaru.

### 3.3.13. Śledzenie Sesji


Używanie sesji w servletach wiąże się z czterem następującymi krokami:
- dostępem do obiektu Session związanym z obecnym żądaniem
- Obiekty sesji są typu HttpSession, jednak są one tylko tablicami hashowanymi, które przetrzymują obiekty użytkownika. Aby uzyskać obiekt HttpSession należy wywołać metodę `getSession()` na obiekcie `HttpServletRequest`.

```java
HttpSession session = request.getSession();
```

Metoda `getSession(true)` tworzy nową sesję, jeżeli sesja nie istnieje dla danego klienta. Natomiast metoda `getSession(false)` zwraca `null` jeżeli sesja nie istnieje.

Należy zadbać, aby metoda `getSession()` była wywoływana przed wysłaniem informacji do klienta.
- wyszukiwaniem informacji związanych z sesją
  Obiekt HttpSession jest przechowywany na serwerze i nie jest przesyłany przez sieć pomiędzy klientem, a serwerem. Ponieważ HttpSession posiada wewnątrz tablicę haszowaną, dlatego może on zawierać dowolną ilość kluczy oraz wartości z nimi związanych. Do znalezienia wartości związanej z danym kluczem należy użyć funkcji `session.getAttribute("key")`, jeżeli nie ma takiego klucza w tablicy zostanie zwrócony `null`. Metoda ta zwraca obiekt typu Object, dlatego należy dokonać rzutowania.

- przechowywanie informacji w sesji
  Aby powiązać jakiś obiekt z sesją należy wykorzystać metodę `setAttribute(key,value)`. Metoda ta zastępuje poprzednią wartość dla podanego klucza. Aby usunąć poprzednią wartość należy posłużyć się metodą `removeAttribute()`.
  - usunąć dane dopisane przez servlet – użycie metody `removeAttribute("key")`,
  - usunąć całą sesję – użycie metody `invalidate()`. Powoduje usunięcie wszystkich danych sesji, wylogowanie użytkownika i usunięcie wszystkich jego sesji – użycie metody `logout`. 
3.4. Przebieg laboratorium

Druga część instrukcji zawiera zadania do praktycznej realizacji, które demonstrują zastosowanie technik z omawianego zagadnienia.

3.4.1. Zadanie 1. Wykorzystanie mechanizmu cookies

Zadanie polega na zaimplementowaniu servletu wykorzystującego mechanizm ciasteczek do identyfikacji użytkownika. Servlet powinien wyświetlać napis „Witaj <imię>” jeśli użytkownik już wcześniej korzystał z servletu. Jeśli robi to po raz pierwszy, powinno zostać wyświetlone zapytanie o imię (formularz HTML), a następnie powinno ono zostać zapisane jako ciasteczek.

3.4.2. Zadanie 2. Dostęp do parametrów żądania servletu

Zadanie polega na stworzeniu aplikacji webowej w skład której będą wchodzić servlet oraz strona HTML. Strona powinna zawierać formularz z polami:
- imię i nazwisko (jako osobne pola),
- adres,
- telefon,
- płeć.

Strona powinna zawierać również przycisk „Akceptuj” do zatwierdzenia danych. Naciśnięcie Akceptuj spowoduje wysłanie parametrów do servletu używając metody POST. Servlet powinien obsługiwać żądanie wyświetlanie nazwy i wartości parametrów w formie tabeli.

3.4.3. Zadanie 3. Servlet z prostą autoryzacją dostępu

Zadanie polega na zaimplementowaniu servletu wykorzystującego podstawową autoryzację użytkownika z użyciem mechanizmu dostępnego w HTTP. Po uruchomieniu servletu powinien on sprawdzić, czy dostęp jest autoryzowany.

Servlet powinien:
2. Jeżeli w żądaniu nie ma takiego nagłówka, to powinno zostać wyświetlone pytanie o login i hasło.
3. Po otrzymaniu nagłówka „Authorization“ zawarte w nim dane powinny być odkodowane (podczas autoryzacji w nagłówku zawarte jest hasło i login zakodowane algorytmem Base64)
4. Sprawdzić poprawność podanych porównując z wczytany plikiem „passwords.properties“. Wykorzystać klasę Properties.
5. Jeżeli pary <login,hasło> będą zgodne, powinna zostać wyświetlona strona o treści „Dostęp przyznany“. W przeciwnym razie powinno ponownie pojawić się zapytanie o login i hasło.

Należy napisać także drugi program w Javie, który powinien:
1. Utworzyć obiekt klasy Properties, w którym zostaną zapisane przykładowe wpisy <klucz=login, wartość=hasło>.
2. Zapisać podane wpisy jako plik „passwords.properties“.
4. Java Server Pages

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące technologii Java Server Pages oraz praktycznych przykładów jej użycia. Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej części laboratorium.

4.1. Tematyka laboratorium


JSP zostało zbudowane na technologii serwerów i można je sobie wyobrazić jako wysoko poziomową abstrakcję serwerów. Strona JSP w procesie translacji jest tłumaczną do serwerów. Składnia JSP jest płynnym połączeniem dwóch typów składni: statycznej składni znaczników HTML oraz dynamicznie wykonywanych bloków kodu Javy. O servletach można powiedzieć że jest to kod znaczników HTML osadzony w kodzie Javy, natomiast w JSP jest odwrotnie. Technologia JSP jest to osadzenie kodu Javy w kodzie strony HTML.

Technologia JSP definiuje wiele standardowych elementów użytecznych dla aplikacji webowych oraz możliwość rozszerzenia składni JSP poprzez implementację własnych elementów technologii JSP. Zbiór powszechnie używanych elementów które rozszerzają JSP jest zdefiniowany w JSP Standard Tag Library (JSTL).

4.2. Cel laboratorium

Celem laboratorium jest wykorzystanie w praktyce najważniejszych elementów technologii JSP. Elementy te są podstawą tworzenia zaawansowanych aplikacji webowych. Podczas realizacji tego laboratorium zapoznasz się z:

- wykorzystywaniem trzech typów elementów stron JSP,
- możliwościami strony JSP,
- łączeniem technologii serwerów i JSP w jednej aplikacji.

Dzięki laboratorium nabędziesz praktycznych umiejętności w tworzeniu warstwy prezentacji aplikacji webowych.
4.3. Zagadnienia do przygotowania

Przed przystąpieniem do realizacji laboratorium należy zapoznać się z zagadnieniami dotyczącymi:
- servletów,
- budową i użyciem pliku web.xml, sun-web.xml.

Literatura:
[2] Dokumentacja na stronach Oracle

4.4. Opis laboratorium

Laboratorium koncentruje się na technologii tworzenia warstwy prezentacji w aplikacjach webowych przy pomocy technologii Java Server Pages. Informacje zawarte w części teoretycznej laboratorium są tylko przeglądamy najważniejszej funkcjonalności i aspektów technologii.

4.4.1. Wstęp do JSP

Aby stworzyć przykładowy kod JSP potrzebujemy wiedzieć co strona ma przedstawiać. Stworzymy stronę JSP, która przedstawia wynik działania poniżej metody w Javie:

```java
import java.util.Date;
public class Time {
    public static void main(String[] args) {
        Date theDate = new Date();
        if (theDate.getHoure() < 12)
            System.out.println("Good morning");
        else
            System.out.println("Good afternoon");
    }
}
```

Taki kod technologii servletów wyglądałby tak:
```
import java.io.*; import java.util.Date;...
@WebServlet("/hellow")
public class hello extends HttpServlet {
    public hello() {
        super();
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        Date theDate = new Date();
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
```

```
if (theDate.getHours() < 12)
    System.out.println("Good morning");
else
    System.out.println("Good afternoon");
System.out.println("</body></html>");

protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {}

A w technologii JSP:
<%@ taglib prefix="c"
uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<body>
<jsp:useBean id="clock" class="java.util.Date"/>
<c:choose>
    <c:when test="${clock.hours < 12}"
    <h1>Good morning</h1>
    </c:when>
    <c:when test="${clock.hours < 18}"
    <h1>Good afternoon</h1>
    </c:when>
</c:choose>
</body>
</html>

Całość składni zostanie omówiona w dalszej części instrukcji. Powyższy przykład służy tylko celom demonstracyjnym. Kod HTML generowany przez servlet i JSP, dla godziny 8 rano, będzie wyglądał następująco:

<html>
<body>
    <h1>Good morning</h1>
</body>
</html>

We wczesnych wersjach JSP istniała możliwość osadzenia kodu Javy w tak zwanych elementach skryptowych. Jest to wygodny sposób na osadzenie kodu, jednak jest on odradzany i wychodzi już z użycia. Kod powyższego programu z wykorzystaniem elementów skryptowych wyglądałby tak:

<%@ taglib prefix="c"
uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<body>
    <%java.util.Date theDate = new java.util.Date();%>
    <% if (theDate.getHours() < 12) { %>

Rozwiązanie z blokami skryptowymi nie jest dobre, ponieważ osadzanie zbyt dużej ilości kodu Javy w stronie webowej nie jest lepsze, niż używanie elementów HTML w programie po stronie serwera i często prowadzi do stworzenia aplikacji trudnej w utrzymaniu i debugowaniu. Przeglądając te kody możemy zauważyć kolejne stopnie abstrakcji od zwykłego kodu Javy, poprzez servlety, bloki skryptowe, a kończąc na stronie JSP bez bloków skryptowych.

Technologii JSP używamy tylko w warstwie prezentacji. Jest ona często nadużywana przez programistów, z powodu możliwości wykonania kodu Javy w stronie JSP. Nie jest to poprawne podejście ponieważ burzy rozdzielenie warstwy prezentacji od logiki biznesowej. JSP pozwala na oddzielenie języka znaczników od języka programowania. Można to osiągnąć poprzez integrację technologii JSP z innym elementem J2EE.

Ponieważ JSP jest zbudowane na Servletach, ma ono dostęp do API narzędzi Enterprise takich jak:

- JDBC – Java Data Base Connector
- RMI i OMG CORBA – Remote Method Invocation
- JNDI – Java Naming and Directory Interface
- EJB – Enterprise JavaBeans
- JMS – Java Message System
- JTA – Java Transcription API
- JAXP/JAXR – Java API for XML Processing/Registering
- JavaMail

4.4.2. Alternatywne rozwiązania

Pozostałe rozwiązania do dynamicznego generowania dokumentów HTML. Rozwiązania różnią się od siebie funkcjonalnością, niektóre rozwiązania mają funkcjonalność, której JSP nie posiada i odwrotnie. Tymi rozwiązaniami są:

- Active Server Pages,
- PHP,
- ColdFusion,
- Java Servlet template engines.

Zalety JSP:

- JSP wspiera obie metody generowania dokumentów HTML: skryptową i bazującą na elementach oraz pozwala programistom na tworzenie bibliotek tagów na użytek specyficznych aplikacji.
- Strony JSP są kompilowane w celu zwiększenia efektywności serwera.
- Strony JSP są łatwo integrowane z innymi technologiami J2EE.
- JSP jest specyfikacją a nie produktem, to oznacza że dowolna osoba lub zespół może opracować swoją własną implementacje specyfikacji.
4.4.3. Problem z Servletami

Technologie jakie jak FastCGI, mod_perl, ISAPI i servlyty Javy cierpią na jeden wspólny problem: generują strony webowe poprzez osadzenie kodu HTML w kodzie programu. Skutkiem takiego rozwiązania jest przeniesienie tworzenia stron webowych tylko i wyłącznie na programistów. Rozwiązanie jakim jest JSP jest zupełnie inne. JSP pozwala na osadzenie bloków kodu Javy, zwanych Scriptletami, w stronach HTML.

Innym problemem jest to, że w wielu aplikacjach opartych na technologii servletów, logika aplikacji, przetwarzanie żądania i generowanie odpowiedzi są obsługiwane przez pojedynczą klasę. Takie podejście jest trudne w obsłudze, modyfikacji i szukaniu błędów. W bardziej strukturalnych aplikacjach, również opartych na servletach, oddzielone zostały elementy takie jak: przetwarzanie żądania, generowanie odpowiedzi, logika aplikacji i umieszczoną w kilku servletach. Jednak i to rozwiązanie ma swoje wady, jak każde oparte na czystej technologii servletów:

- W celu zarządzania dowolnym aspektem aplikacji potrzebna jest gruntowna wiedza programistyczna o Javie, ponieważ kod wykonywany i kod HTML są pomieszane ze sobą.
- W celu zmiany wyglądu aplikacji, jej zachowania albo dodania nowej funkcjonalności potrzebna jest aktualizacja kodu i ponowna kompilacja.
- Trudno jest korzystać z zalet narzędzi do zarządzania stronami webowymi podczas projektowania interfejsów aplikacji, np. edytor stron WYSIWYG. Jeśli takie narzędzie jest używane do tworzenia wyglądu strony, kod HTML musi być ręcznie osadzony w serwlecie, co może powodować wiele błędów i zabierać dużo czasu.

Użycie technologii JSP w aplikacjach webowych pozwoli rozwiązać większość problemów poprzez rozdzielenie obsługi żądania i logiki aplikacji od warstwy prezentacji. Przykładowe rozdzielenie może polegać na: osadzeniu statycznego kodu HTML i kilku elementów do generowania dynamicznych części strony w JSP. Obsługa żądania może zostać umieszczona w serwlecie, a logika aplikacji w ziarnach EJB.

4.4.4. Anatomia strony JSP

Strona JSP to zwyczajna strona webowa z elementami JSP umieszczonymi w kodzie, w celu dynamicznego generowania części strony. Na stronie JSP wszystko to co nie jest elementem JSP nazywamy tekstem szablonowym (ang. template text). Takim tekstem może być jakikolwiek tekst: HTML, XML, WML albo zwykły tekst. Poniżej znajduje się przykładowy kod źródłowy strony z rozdzieleniem na elementy JSP i template text:

```jsp
<%@ page contentType="text/html" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core1" %>
<html>
<head>
<title>JSP example 1</title>
</head>
<body>
```
4.4.5. Przetwarzanie strony JSP


Żądanie o stronę JSP, która przeszła już fazę translacji i która nie została zmodyfikowana po tej fazie, zostaje od razu przekazane do fazy przetwarzania. Po modyfikacji strona musi przejść fazę translacji ponownie. Kontener JSP często jest implementowany jako servlet, który jest zdolny do obsługi stron JSP. Częstą praktyką jest dostarczanie kontenera JSP i kontenera servletów jako jednej paczki pod nazwą web container. Poza fazą translacji strona JSP jest przetwarzana dokładnie tak, jak servlet.


Dyrektwy zawierają informacje o stronie JSP które nie ulegają zmianie w kolejnych żądaniach. Zawarte są informacje takie jak: nazwa strony, wielkość bufora, informacje o śledzeniu sesji, pliki lub biblioteki, które będą zaimportowane w fazie translacji. Poniższa tabela zawiera elementy o charakterze dyrektyw:

<table>
<thead>
<tr>
<th>Element</th>
<th>Opis</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;%@ page … %&gt;</code></td>
<td>definiuje atrybuty zależne od konkretnej strony</td>
</tr>
<tr>
<td><code>&lt;%@ include … %&gt;</code></td>
<td>pliki które będą zaimportowane w fazie translacji</td>
</tr>
<tr>
<td><code>&lt;%@ taglib … %&gt;</code></td>
<td>deklaracja biblioteki tagów</td>
</tr>
</tbody>
</table>
Elementy akcji wykonują czynności bazując na informacjach, które są dostępne w czasie odebrania żądania o stronę JSP, np. żądanie o czas. Po odebraniu żądania o stronę JSP zawierającą aktualny czas, pobierany jest czas systemowy, umieszczany w kodzie strony HTML i strona zostaje odesłana w odpowiedzi.

Elementy skryptowe pozwalają na dodawanie bloków kodu Javy do strony JSP. Podobnie jak elementy akcji, elementy skryptowe są wykonywane w czasie otrzymania żądania. Mogą one zaimplementować w sobie całą logikę aplikacji, z tego względu należy ich używać z wielką ostrożnością, gdyż aplikacja może okazać się zbyt trudna w późniejszej modyfikacji. Tabela poniżej zawiera elementy o charakterze skryptowych:

<table>
<thead>
<tr>
<th>Element</th>
<th>Opis</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;% … %&gt;</td>
<td>scriptlet używany do osadzenia dowolnego kawałku kodu</td>
</tr>
<tr>
<td>&lt;%= … %&gt;</td>
<td>znaczniki używane do osadzenia wyrażenia, którego wynik powinien zostać dodany do odpowiedzi</td>
</tr>
<tr>
<td>&lt;%! … %&gt;</td>
<td>znacznik używany do deklaracji zmiennych i metod w kodzie strony JSP</td>
</tr>
</tbody>
</table>

Expression Language to język wykorzystywany w celu uzyskania dostępu do żądanych danych i udostępnienia danych poprzez klasy aplikacji. Wyrażenia EL mogą zostać użyte wprost w template text, albo mogą zostać przypisane jako wartości w atrybutach elementów akcji. Składnia jest podobna do języka JavaScript. Zazwyczaj jest on używany do spajania elementów akcji z innymi komponentami.

### 4.4.6. Tworzenie i zarządzanie stronami JSP

Strony JSP umieszczane na serwerze powinny mieć rozszerzenie .jsp. Jest to ważne, gdyż stanowi jedyną wskazówkę dla serwera, że ten plik powinien być przetwarzany przez kontener JSP.

Rozpoczęcie pracy z technologią JSP zaczniemy od uruchomienia przykładowej strony JSP na serwerze. Przykładowa strona JSP znajduje się w folderze ze źródłami i nazywa się przykład.jsp. W dowolnym IDE z podpiętym serwerem tworzymy projekt webowy (np. w NetBeansie Web Application). Kolejno w menu tworzenia projektu wybieramy serwer aplikacji. W katalogu Web Pages umieszczamy plik przykładowy i uruchamiamy projekt. Powinniśmy jako rezultat otrzymać stronę webową, której zawartością będzie tekst:

```
JSP example 1 ...
1 - 2 + 3 % 4 * 5 / 6 = 1.5
```

Pierwsze dwa wiersze strony przykładowej są dyrektywami. Są to elementy, które opisują stronę w aspektach takich jak zawartość strony (MIME), bufor potrzebny na stronę, deklaracja wykorzystywanych zasobów, jak powinny zostać obsłużone błędy wykonania etc. Dyrektywy nie wpływają bezpośrednio na zawartość odpowiedzi przesyłanej do przeglądarki. Instruują one kontener JSP jak powinien potraktować stronę.
Dyrektywy

Strony JSP zazwyczaj rozpoczyna dyrektywa określająca zawartość dokumentu:
<%@ page contentType="text/html" %>

Dyrektywy składają się z nawiasu otwierającego (<%@), nazwy dyrektywy (page), atrybutów para nazwa/wartość (contentType="text/html") i kończą się nawiasem zamykającym (>%). Nazwa i wartość atrybutów są czułe na wielkość znaków, a wszystkie wartości muszą być otoczone pojedynczym lub podwójnym cudzysłowem.

Dyrektywa page posiada wiele typów atrybutów. W przykładzie zawiera atrybut contentType określa on zawartość strony w typach MIME. Jeśli pominie się tę dyrektywę kontener domyślnie będzie traktował stronę jako text/html.

Inne typy atrybutów to errorPage, isErrorPage, session, buffer, pageEncoding i autoFlush. Istnieją inne typy atrybutów ale są rzadko wykorzystywane.

Akcje

Elementy akcji w naszym przykładzie jedynie wykonują proste wyrażenie arytmetyczne. Jednak mogą wykonywać szereg innych różnych zadań takich jak obsługa plików, wysyłanie maili, obsługa bazy danych itd. W powyższym przykładzie elementem akcji jest linia:

```jsp
<c:out value="${1 - 2 + 3 % 4 * 5 / 6}" />
```

Składnia elementów akcji w JSP jest podobna do znaczników HTML. Składnia elementu akcji, który posiada ciało, jest złożona z otwierającego tagu, z możliwymi atrybutami, ciała funkcji, i tagu zamykającego:

```jsp
<prefix:action name atrybut_1="wartość", … , atrybut_n="wartość">
    ciało funkcji
</prefix:action_name>
```

Ciało funkcji może zawierać kolejne wyrażenia do wykonania, albo może zawierać inne elementy akcji. W przypadku kiedy element akcji nie zawiera w sobie ciała funkcji składnia jest nieco uproszczona:

```jsp
<prefix:action_name atrybut_1="wartość", … , atrybut_n="wartość"/>
```

Elementy akcji i tagi są zgrupowane w bibliotekach, nazywanych bibliotekami tagów (ang. tag libraries). W dyrektywie określającej bibliotekę tagów, określamy prefix używany przy wywoływaniu elementów akcji (taglib prefix="c"). Określa on przestrzeń nazw dla tej biblioteki tagów. W jednej stronie JSP możemy mieć kilka różnych przestrzeni nazw.

4.4.7. Architektura i Używanie biblioteki JSTL

JSTL stała się biblioteką na tyle potrzebną i powszechnie używaną, że weszła do formalnej specyfikacji JSP. To pozwala różnym zespołom programistycznym na implementowanie oficjalnej specyfikacji biblioteki JSTL na własny sposób, zoptymalizowany pod kątem kontenera JSP.
Biblioteka JSTL zawiera w sobie pięć różnych bibliotek tagów. Poniższa tabela zawiera nazwę biblioteki, domyślne URI dla niej i preferowany prefiks.

<table>
<thead>
<tr>
<th>Biblioteka</th>
<th>URI</th>
<th>Prefiks</th>
</tr>
</thead>
<tbody>
<tr>
<td>core</td>
<td><a href="http://java.sun.com/jsp/jstl/core">http://java.sun.com/jsp/jstl/core</a></td>
<td>c</td>
</tr>
<tr>
<td>fmt</td>
<td><a href="http://java.sun.com/jsp/jstl/fmt">http://java.sun.com/jsp/jstl/fmt</a></td>
<td>fmt</td>
</tr>
<tr>
<td>sql</td>
<td><a href="http://java.sun.com/jsp/jstl/sql">http://java.sun.com/jsp/jstl/sql</a></td>
<td>sql</td>
</tr>
<tr>
<td>XML</td>
<td><a href="http://java.sun.com/jsp/jstl/xml">http://java.sun.com/jsp/jstl/xml</a></td>
<td>x</td>
</tr>
<tr>
<td>functions</td>
<td><a href="http://java.sun.com/jsp/jstl/functions">http://java.sun.com/jsp/jstl/functions</a></td>
<td>fn</td>
</tr>
</tbody>
</table>

Biblioteka Core zawiera elementy akcji odpowiedzialne za kontrolę przepływu, mapowanie URI, importowanie zasobów, łapanie wyjątków, wyrażenia warunkowe, pętle, dopisywanie do odpowiedzi różnych elementów i tym podobne akcje ogólnego przeznaczenia.

Biblioteka XML processing zawiera elementy akcji o charakterze przetwarzania dokumentów XML. Część akcji pokrywa się z akcjami z biblioteki Core, jednak są akcje typowe dla przetwarzania plików XML takie jak parsowanie albo transformacje XML.

Biblioteka I18N formatting, w skrócie fmt, służy do internacjonalizacji i formatowania tekstu. Znajdują się w niej akcje służące do ustawiania typu kodowania znaków, lokalizacji oraz strefy czasowej, formatowania i parsowania daty oraz numerów.

Biblioteka Database access służy do obsługi bazy danych przy pomocy języka SQL. Zawiera elementy umożliwiające wykonanie na bazie danych wyrażeń typu SELECT i zapis wyników we wskazanej zmiennej, zmiany, dodawania i usuwania wpisów w bazie danych oraz wywołanie wyrażenia commit albo rollback.

Biblioteka Functions zawiera elementy o charakterze ogólnego przeznaczenia służące głównie do pracy z tekstem: szukanie podciągu w łańcuchu znaków, długości tekstu albo ilości elementów w kolekcji, sprawdzanie jaki jest indeks elementu w tabeli, dzielenie, łączenie, zamiany, elementów w tekście, operacje na wielkości liter, usuwanie białych znaków.

Używanie elementów z biblioteki JSTL jest takie jak w przypadku używania zwykłych elementów akcji. Deklaracja biblioteki a następnie w znacznikach prefiks: nazwa elementu i atrybuty oraz ciało funkcji, jeśli takie istnieje.

### 4.4.8. Współpraca JSP i Servletów

Tak jak zostało to zaznaczone we wstępie kombinacja JSP i servletów czysto oddziela od siebie logikę aplikacji od warstwy prezentacji aplikacji. Specyfikacja JSP zaleca dwa podejścia do rozdzielenia tych warstw. W podejściu pierwszym strona JSP jest odpowiedzialna za obsługę żądania i generację odpowiedzi.

W drugim podejściu integruje się użycie i servletów i JSP. Umożliwia to czyste wykorzystanie modelu Model-View-Controller, gdzie servlet spełnia rolę Controllera, JSP rolę View a za Model jest odpowiedzialna technologia JavaBean.

Przekazywanie żądania następuje poprzez klasę RequestDispatcher. Aby uzyskać RequestDispatcher powiązany z konkretnym adresem URL, należy wywołać metodę
getRequestDispatcher na adresie URL. Na przykład uzyskanie Dispatchera powiązanego ze stroną index.jsp można uzyskać za pomocą kodu:

```java
RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/index.jsp");
```

Kiedy już uzyskamy takie powiązanie należy użyć metody forward do zakończenia przekazywania żądania. Metoda forward wymaga dwóch argumentów HttpServletRequest i HttpServletResponse. Przykładowy kod przekazania sterowania może wyglądać tak:

```java
RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/index.jsp");
dispatcher.forward(request, response);
```

Mimo, że w większości działających aplikacji przekazywanie żądania jest od servletu do JSP, to możliwe jest też przekazywanie w drugą stronę. W tym celu należy użyć elementu `<jsp:forward page="URL">`. Możliwe jest też ciekawe połączenie przekazywania żądania w zależności od wyrażeń warunkowych. Przykładowy kod który nie wymaga szerszego komentarza wygląda tak:

```jsp
<% String destination;
   if (Math.random() > 0.5)
      destination = "/examples/page1.jsp";
   else
      destination = "/examples/page2.jsp";
%
<jsp:forward page="<%= destination %>" />
```

Są dwa główne miejsca, gdzie można składować dane, które zostaną użyte na stronie JSP. W HttpServletRequest i w ziarnie umieszczonym w miejscu podanym jako argument jsp:useBean.

### 4.4.9. Prekompilacja strony JSP

 Ważną korzyścią korzystania z JSP jest to, że strona jest zawsze najpierw kompilowana przed użyciem jej przez serwer. Strona jest kompilowana do poziomu kodu wykonywalnego przy pierwszym żądaniu strony i wywołuje kod wynikowy przy każdym kolejnym żądaniu.

 W celu uniknęcia dłuższego wczytywania strony JSP przy pierwszym jej użyciu, możemy skorzystać z narzędzia prekompilacji wszystkich stron JSP. Przy mniejszych stronach nie zauważymy większej różnicy, ale przy mocno rozbudowanych stronach zawierających mnóstwo elementów akcji czas będzie znacznie krótszy. Inną zaletą użycia stron prekompilowanych jest uniemocnienie ich zmian po instalacji strony JSP na serwerze. W tym przypadku można po prostu prekompilować wszystkie strony, zdefiniować mapowanie URL na stronę JSP i zainstalować aplikację na serwerze.

 Jedynym sposobem na prekompilację jest uruchomienie wszystkich stron w IDE i skopiowanie plików klas z całą resztą aplikacji na serwer. Innym sposobem jest wykorzystanie specjalnego argumentu wywołania, który został umieszczony w specyfikacji JSP. Parametr ten daje wskazówkę kontenerowi JSP, że strona powinna
zostać skompilowana bez obsługi żądania. Zaletą takiego rozwiązania jest możliwość jego automatyzacji, używając jakiegoś prostego narzędzia typu load-test z obsługą podstawowych parametrów wywołania. Ponieważ w tym sposobie strony nie są wykonywane, a tylko prekompilowane, logika aplikacji, która wymaga wywołania stron JSP w określonym porządku, nie wpływa na kompilację. Możliwymi wartościami parametru jsp_precompile są true, false, lub brak wartości.

Wartości true i brak wartości oznaczają konieczność prekompilacji strony. Wartość false nie jest zbyt użytoczna, gdyż serwer zignoruje ten parametr.

Po procesie prekompilacji można spakować aplikację do pliku JAR nie używając plików stron JSP, a wygenerowanych plików klas servletów. Można to zrobić poprzez dodawanie definicji mapowania URL w pliku deskryptora deployingu (web.xml).

Posłużmy się przykładem. Mamy fragment pliku deskryptora deployingu:

```xml
<web-app>
   ...
   <servlet>
      <servlet-name>easy</servlet-name>
      <servlet-class>com.soajsp.ch5.easy_jsp</servlet-class>
   </servlet>
   ...
   <servlet-mapping>
      <servlet-name>easy</servlet-name>
      <url-pattern>/ch5/easy.jsp</url-pattern>
   </servlet-mapping>
   ...
</web-app>
```

Element <servlet> mapuje pełną nazwę klasy servletu na symboliczną nazwę. Element <servlet-mapping> wskazuje kontenerowi JSP gdzie ma szukać klasy odpowiedzialnej za stronę JSP w chwili wywołania strony.

Mechanizm prekompilacji strony JSP i ustawienie mapowania URL ma dwie główne zalety. Pierwszą jest fakt, że przy użyciu tego mechanizmu odpowiedzi strony JSP są szybsze, ponieważ kontener JSP nie musi przejść przez całą stronę JSP w celu ustalenia, którego servlet użyć. Kolejną zaletą jest fakt, że tylko twórca może załączyć strony JSP do aplikacji, co implikuje większą kontrolę i bezpieczeństwo aplikacji opartej na stronach JSP.

Wykonanie prekompilacji w ten sposób nie jest trudnym elementem tworzenia aplikacji, ale jest pracochłonne i żmudne. Jednak narzędzie Ant połączone z JspC może wykonać całą pracę za nas.
4.5. Przebieg laboratoryum

Druga część instrukcji zawiera zadania do praktycznej realizacji, które demonstrują zastosowanie technik z omawianego zagadnienia.

4.5.1. Zadanie 1. Elementy JSP w akcji

Zadanie polega na praktycznym wykorzystaniu wiedzy z części teoretycznej. Należy wykonać kolejno:
1. Utworzyć projekt webowy i na stronie domyślnie stworzonej przez IDE sprawdzić poprawność działania środowiska.
2. Stworzyć kolejną stronę JSP, która będzie zawierać elementy:
   a. elementy HTML – wypisanie krótkiego tekstu
   b. element JSTL – w pętli wypisanie liczb 1-10
   c. element skryptowy – wyrażenie warunkowe: jeśli jest przed południem – wypisz „GoodMorning”, jeśli po „GoodAfternoon”
   d. element Expression Language – dowolne wyrażenie arytmetyczne
3. Przekazać sterowanie ze strony domyślnej na stronę JSP i uruchomić aplikację
4. Sprawdzić poprawność aplikacji poprzez wywołanie domyślnej strony
5. Zmienić kod tak, aby strona nowo stworzona była dołączana do strony domyślnej

4.5.2. Zadanie 2. Własne elementy akcji

Zadanie polega na stworzeniu dwóch własnych elementów akcji i użyciu ich na stronie JSP. Pierwszy element ma za zadanie wypisać podaną ilość liczb pseudolosowych. Drugi ma za zadanie dla podanego jako argument tekstu zamienić wszystkie litery na duże litery i jeśli argument length jest true, wypisać długość tekstu, a następnie użyć tych elementów na dowolnej stronie JSP.

4.5.3. Zadanie 3. JSP i servlyte

Zadanie polega na napisaniu dwóch modeli aplikacji. Schemat poglądowy działania pierwszej aplikacji:
1. Żądanie odbiera servlet i w zależności od parametrów żądania podejmuje odpowiednie akcje.
   a. Count – możliwe wartości 0, 1, 2. Dla innych wysyła błąd 404. Dla 0 odsyła napis „Nie mam nic dla ciebie”. Dla 1 w odpowiedzi odsyła stronę JSP pokazującą datę i godzinę. Dla 2 odsyła stronę z danymi o połączeniu (numer IP portu)
2. Żądanie odbiera JSP i w zależności od godziny wywołuje jeden z dwóch serwletów. Dla parzystej minuty servlet 1, dla nieparzystej servlet 2.
   a. Servlet 1 – tworzy odpowiedź, która zawiera tekst (imię nazwisko przedmiot)
   b. Servlet 2 – tworzy odpowiedź która zawiera dane o połączeniu
4.6. Opracowanie i sprawozdanie

Realizacja laboratorium pt. „JSP JavaServer Pages” polega na wykonaniu wszystkich zadań programistycznych podanych w drugiej części tej instrukcji. Wynikiem wykonania powinno być sprawozdanie w formie wydruku papierowego. Sprawozdanie powinno zawierać:

- opis metodyki realizacji zadań (system operacyjny, język programowania, biblioteki, itp.),
- algorytmy wykorzystane w zadaniach (związane, jeśli zastosowane zostały rozwiązania nietypowe),
- opisy napisanych programów wraz z opcjami,
- uwagi oceniające ćwiczenie: trudne/łatwe, nie/wymagające wcześniejszej znajomości zagadnień (wymienić jakich),
- wskazówki dotyczące ewentualnej poprawy instrukcji celem lepszego zrozumienia sensu oraz treści zadań.
5. XML i SOAP

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące języka XML, schematów XML i protokołu SOAP oraz wykorzystania ich do zdalnego wywoływania obiektów i przesyłania danych. Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej części laboratorium.

5.1. Tematyka laboratorium

Tematyką laboratorium jest programowanie aplikacji klient-serwer w oparciu o protokół SOAP. SOAP (ang. Simple Object Access Protocol) jest protokołem warstwy aplikacji modelu ISO/OSI standaryzowanym przez organizację W3C.

Protokół SOAP został opracowany w celu sprawnej komunikacji między aplikacjami w sieci Internet. Jest prostym i rozszerzalnym protokołem komunikacyjnym, niezależnym od platformy i języka programowania. SOAP umożliwia zdalny dostęp do obiektów (porównaj: RPC, CORBA, DCOM, DCOP, IPC, D-Bus). Wykorzystuje XML do kodowania wywołań, natomiast do ich przenoszenia najczęściej wykorzystywany jest protokół HTTP (możliwe jest też wykorzystanie protokołu RPC, SMTP i innych).

Komunikacja za pomocą protokołu SOAP odbywa się wg schematu zgłoszenie – odpowiedź (podobnie jak pobieranie stron internetowych za pomocą protokołu HTTP). Zgłoszenie i odpowiedź nazywamy komunikatem. Komunikat to dokument XML zbudowany wg poniższego schematu:

- **koperta** – określa szkielet opisujący, co znajduje się w komunikacie i jak go przetwarza:
  - **nagłówek** – zbiór reguł kodujących potrzebnych do rozsyfrowania typów i struktur danych zdefiniowanych wewnątrz aplikacji,
  - **treść** – reguły (obiekt, metoda, parametry) dotyczące wywołania zdalnych metod i odczytu odpowiedzi.

5.2. Cel laboratorium

Celem laboratorium jest poznanie podstawowych własności języka XML i protokołu SOAP oraz ich praktycznego jego zastosowania. Podczas tego laboratorium zostanie:

1. Zbudowana prosta aplikacja w języku Java zapisująca dane do pliku XML oraz wczytująca dane z wcześniejszą walidacją poprawności dokumentu na podstawie pliku schematu XML.
5.3. Zagadnienia do przygotowania

Przed przystąpieniem do realizacji laboratorium należy zapoznać się z zagadnieniami dotyczącymi:

- budowy modelu ISO/OSI,
- protokołu HTTP,
- języka XML [1],
- protokołu SOAP [2],
- obsługa programu tcpdump.


5.4. Opis laboratorium

5.4.1. Język XML

Język XML służy do przechowywania danych w formacie tekstowym i strukturze drzewa (struktura hierarchiczna). Język XML jest niezależny od platformy i języka programowania, co umożliwia wymianę dokumentów między heterogenicznymi systemami. Przykładowy plik XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="books" type="books"/>

  <xs:complexType name="books">
    <xs:sequence>
      <xs:element name="book" type="book" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="book">
    <xs:sequence>
      <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
      <xs:element name="title" type="title" maxOccurs="unbounded"/>
      <xs:element name="isbn" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="title">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="lang" type="xs:string" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:schema>
```
Poprawny plik XML składa się z nagłówka określającego wersję języka XML zastosowanego w pliku i kodowania znaków. Atrybut standalone="yes" jest nieobowiązkowy i informuje, iż plik ten nie zawiera powiązań z innymi plikami.

Struktura danych w pliku XML składa się z elementów (ang. elements), które można porównać do gałęzi i liści w strukturze drzewa.


Elementy mogą być zagnieżdżone w sobie i występować w dowolnej ilości. Wyjątkiem jest element główny (ang. root), który musi występować tylko w jednym egzemplarzu.


Elementy mogą zawierać atrybuty (ang. attributes). Definiowane są one w znaczniku początku elementu. Np. znacznik <title lang="pl"> zawiera jeden atrybut o nazwie „lang” i wartości „pl”. Nazwy atrybutów nie mogą się powtarzać w obrębie jednego znacznika.

Dane tekstowe i wartość atrybutów nie mogą zawierać znaków początku i końca znaczników. Znak mniejszości (<) należy zastąpić wyrażeniem &lt; a znak większości (>) wyrażeniem &gt;. Jeżeli potrzebujemy zapisać znak ampersand (&), stosujemy wyrażenie &amp;.

Nazwy znaczników i nazwy atrybutów mogą składać się z liter, cyfr oraz znaków „−” (minus), „_” (podkreślenie) i „.” (kropka). Nazwy nie mogą rozpoczynać się od cyfr, kropki ani minusa. Nie mogą też zaczynać się od „xml” (wielkość znaków dowolna).

Dokument XML może zawierać komentarze. Umieszcza się je pomiędzy wyrażeniami <!-- i -->.

Dokument może zawierać sekcję danych znakowych, która nie będzie przetwarzana przez analizator składni. Wyrażeniem rozpoczynającym taką sekcję jest <! [CDATA[, a wyrażeniem kończącym: ]]>). Sekcja może posłużyć do zagnieżdżenia tekstu zawierającego znaki mniejszości i większości bez zamiany ich odpowiednio na &lt; i &gt;.

O dokumencie XML mówimy, że jest poprawny składniowo (ang. well-formed), jeżeli jest zgodny z regułami składni XML, tj. wszystkie znaczniki i ich atrybuty są poprawnie domknięte i nie zawierają znaków specjalnych w nazwie.

5.4.2. Schematy XML

Schemat XML opisuje strukturę dokumentu XML. Określa jakie znaczniki mogą zostać użyte w dokumencie i z jakimi parametrami. Określa też jakie elementy w jakich elementach mogą zostać zagnieżdzone i w jakiej ilości.


```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="books" type="books"/>
  <xs:complexType name="books">
    <xs:sequence>
      <xs:element name="book" type="book" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="book">
    <xs:sequence>
      <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
      <xs:element name="title" type="title" maxOccurs="unbounded"/>
      <xs:element name="isbn" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="title">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="lang" type="xs:string" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:schema>
```

Powyższy plik odnosi się do pliku XML z przykładu zamieszczonego w poprzednim podrozdziale. Określa on, że element główny ma nazwę „books”, który jest elementem złożonym (może zawierać elementy zagnieżdżone) i składa się
z dowolnej ilości (od zera do nieskończoności: minOccurs="0" maxOccurs="unbounded") elementów o nazwie „book”.

Element „book” również jest elementem złożonym i składa się z co najmniej jednego elementu prostego „author” (minOccurs i maxOccurs domyślnie przyjmują wartość 1), dokładnie jednego elementu „isbn” i co najmniej jednego elementu złożonego „title”.

Element „title” co prawda przechowuje wartość prostą, ale posiada atrybut „lang”, którego użycie jest wymagane (use="required").

Dzięki dokumentom schematu XML możliwe jest wykorzystanie gotowych narzędzi do wstępnego zweryfikowania poprawności formatu danych w dokumencie.

5.4.3. Protokół SOAP

SOAP jest protokołem służącym do komunikacji między aplikacjami sieciowymi przez sieć rozległą WAN (np. Internet). Jest on protokołem niezależnym od platformy i języka programowania. Do kodowania wiadomości i ich przesyłu wykorzystuje istniejące technologie – odpowiednio XML i HTTP.

Jednymi z najistotniejszych własności protokołu SOAP są:

- Prosty w implementacji protokół zaprojektowany z myślą o komunikacji w Internecie z zastosowaniem istniejących technologii do komunikacji i kodowania danych.
- Do kodowania wykorzystywany jest język XML, który jest niezależny od języka programowania i architektury procesora. Ponadto dzięki wysokiej popularności języka XML, dostępna jest pełna gama bibliotek i narzędzi do przetwarzania dokumentów XML.
- Do komunikacji wykorzystywany jest najczęściej protokół HTTP/HTTPS, który jest protokołem bardzo popularnym w sieci Internet. Aplikacja kliencka może łączyć się z odległym serwerem bez konieczności zmiany konfiguracji firewalla, jeżeli wcześniej jego ustawienia umożliwiały użytkownikowi na przeglądanie stron internetowych. Ponadto zastosowanie protokołu HTTPS zapewnia bezpieczeństwo przesyłanych danych.
- Protokoły HTTP i HTTPS nie są jedynymi protokołami, które mogą posłużyć jako medium komunikacyjne. Za medium może posłużyć praktycznie każda technologia pozwalająca na przesyłanie danych. Z praktycznych przykładów można wymienić: JMS, SMTP i RPC.
- Minusem jest jednak często duży narzut komunikacyjny na rozmiar danych przesyłanych między aplikacjami. W szczególności nie nadaje się do przesyłania dużej ilości danych binarnych – rozwiązaniem tego problemu jest zastosowanie protokołu SwA (SOAP with Attachments).

Przedstawiony poniżej kod dokumentu SOAP przedstawia zlecenie wykonania funkcji GetStockPrice z parametrem StockName o wartości „IBM”.

```xml
<?xml version="1.0"?>
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299

POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299

<?xml version="1.0"?>
```
5.4.4. Język XML w Java

Język Java wraz ze środowiskiem uruchomieniowym dostarczonym przez Oracle (dawniej Sun) dostarcza biblioteki ułatwiające parsowanie, edycję i walidację dokumentów XML. Do bardziej zaawansowanych zastosowań dostępna jest biblioteka JAXB, która umożliwia zapis stanu całych obiektów odpowiednio wcześniej przygotowanych klas do pliku XML i odtworzenie ich stanu.

W standardowej bibliotece Java SE sposób reprezentacji dokumentów XML odbywa się za pomocą modelu obiektowego DOM (Document Object Model).

Aby stworzyć plik DOM potrzebujemy najpierw uzyskać obiekt klasy fabryki budowniczych dokumentów (klasa DocumentBuilderFactory dostępna w pakiecie javax.xml.parsers). Obiekt wspomnianej klasy uzyskiwamy za pomocą instrukcji:

```java
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
```

W obiekcie klasy DocumentBuilderFactory można dokonać dodatkowych ustawień takich jak:

- `dbf.setNamespaceAware(true);` – przetwarzanie przestrzeni nazw,
- `dbf.setValidating(true);` – walidacja w oparciu o plik schematu XML.

Jeżeli decydujemy się na skorzystanie z walidacji, a schemat XML został zapisany w formacie pliku XML Schema, musimy o tym odpowiednio powiadomić obiekt fabrykę:

```java
```

Następnie musimy utworzyć obiekt budowniczego dokumentów (klasa DocumentBuilder z pakietu javax.xml.parsers):

```java
DocumentBuilder db = dbf.newDocumentBuilder();
```

Obiekt klasy budowniczego dokumentów umożliwia nam utworzenie nowego, pustego dokumentu (metoda `newDocument()`) lub wczytanie istniejącego (metoda `parse(File plikXml)`). Metoda `parse` występuje w kilku wariantach, m.in. z możliwością podania adresu w postaci URI lub obiektu klasy `InputStream`. 
Wartością zwracaną obu metod jest obiekt klasy Document z pakietu org.w3c.dom.

Nowe elementy tworzy się za pomocą metody createElement(String name) wykonanej na obiekcie klasy Document, która tworzy obiekt klasy Element z pakietu org.w3c.dom. Stworzony element nie zostaje jednak automatycznie dodany do dokumentu. Klasy Element i Document dziedziczą po klasie Node z pakietu org.w3c.dom, która z kolei udostępnia takie metody jak:

- appendChild(Node node) – dodanie elementu,
- removeChild(Node node) – usunięcie elementu,
- getChildNodes() – pobranie listy elementów,
- getTextContent() – pobranie danych tekstowych zapisanych w elemencie
- setTextContent(String text) – wprowadzenie danych tekstowych do elementu.

Natomiast klasa Element implementuje dodatkowo metody:

- getTagName() – pobranie nazwy znacznika,
- getAttribute(String attrName) – pobranie wartości atrybutu o nazwie attrName,
- setAttribute(String attrName, String value) – ustawienie wartości atrybutu,
- hasAttribute(String attrName) – sprawdzenie czy wartość atrybutu została wprowadzona,
- removeAttribute(String attrName) – usunięcie atrybutu,
- getElementsByTagName(String name) – pobranie listy elementów o podanej nazwie.

Należy pamiętać, że wszystkie elementy muszą być gdzieś umieszczone, a najwyższy w hierarchii element należy umieścić w obiekcie klasy Document za pomocą metody appendChild.

Powyższe informacje są wystarczające do wczytania i walidacji pliku XML. Do wyeksportowania modelu DOM do pliku XML potrzebna jest jeszcze klasa Transformer z pakietu javax.xml.transform wraz z klasami pomocniczymi. Do uzyskania obiektu klasy Transformer potrzebujemy wcześniej uzyskać obiekt fabryki TransformerFactory w następujący sposób:

```java
TransformerFactory tf = TransformerFactory.newInstance();
```

Dopiero z obiektu fabryki możemy uzyskać obiekt klasy Transformer:

```java
Transformer t = tf.newTransformer();
```

W obiekcie klasy Transformer trzeba jeszcze ustawić format dokumentu wyjściowego, w naszym przypadku XML:

```java
t.setOutputProperty(OutputKeys.METHOD, "xml");
```

Jeżeli chcemy aby wygenerowany plik XML nie był w jednej linii tylko w formacie bardziej czytelnym dla człowieka, możemy ustawić parametr:

```java
t.setOutputProperty(OutputKeys.INDENT,"yes");
```
Potrzebny jest nam jeszcze obiekt klasy `Source` z pakietu `javax.xml.transform`, który otrzymamy tworząc obiekt klasy po niej dziedziczącej `DOMSource` z pakietu `javax.xml.transform.dom`:

```java
Source source = new DOMSource(doc);
```

Gdzie `doc` to obiekt klasy `Document`. W zapisie wyniku pośredniczy obiekt klasy `Result` z pakietu `javax.xml.transform`, który również uzyskamy tworząc obiekt klasy po niej dziedziczącej np. `StreamResult` z pakietu `javax.xml.transform.stream`:

```java
Result result = new StreamResult(System.out);
```

W tym przypadku wynik zostanie wypisany na standardowe wyjście. Jednak aby się to stało trzeba jeszcze wykonać instrukcję:

```java
t.transform(source, result);
```

Idea obiektów POJO (Plain Old Java Object) narodziła się z konieczności maksymalnego uproszczenia sposobu zapisu stanu obiektów i odtwarzania ich w innym systemie. Obiekty POJO są obiektami zwykłych klas Java. Jedyne co wyróżnia klasy ich obiektów to odpowiednie adnotacje, które informują jakie dane za pomocą jakich getterów powinny zostać pobrane z obiektu oraz za pomocą jakich setterów powinny zostać wprowadzone do nowo utworzonego obiektu tej klasy w niekoniecznie tym samym systemie i niekoniecznie zgodnej binarnie.

W Java do importu i eksportu stanu obiektów POJO do plików XML zastosujemy bibliotekę JAXB.

Aby klasa była zgodna z POJO musi spełniać następujące założenia:
1. Musi posiadać bezargumentowy publiczny konstruktor.
2. Biblioteka JAXB automatycznie wyszukuje właściwości oraz właściwości listowych do składowania w XML na podstawie publicznych właściwości klasy lub publicznych getterów i setterów o nazwach zgodnych z koncepcją:

```java
public Klasa getWłaściwość();
public void setWłaściwość(Klasa value);
```

Natomiast dla list musi zostać zachowana koncepcja:

```java
public List<Klasa> właściwość;
```

lub w przypadku getterów i setterów:

```java
public List<Klasa> getWłaściwość();
public void setWłaściwość(List<Klasa> value);
```

Za pomocą odpowiednich adnotacji można włączyć do przechowywania właściwość prywatną lub wyłączyć właściwość publiczną. Adnotacje umieszczana się przed właściwością klasy (jeżeli dostęp do niej ma być bezpośredni) lub przed getterem (jeżeli dostęp do niej odbywa się za pośrednictwem getterów i setterów).
3. Biblioteka JAXB wyszukuje wszystkie właściwości spełniające ww. reguły. Jeżeli jakieś właściwości nie chcemy przechowywać w XML musimy opatrzeć ją adnotacją:

```java
@XmlTransient
```

4. Jeżeli jakaś właściwość nie jest publiczna, a chcemy ją przechowywać w XML, musimy opatrzeć ją adnotacją `@XmlValue`, `@XmlAttribute` albo `@XmlElement`.

5. Adnotacja `@XmlValue` jest bezparametrowa. Właściwość opatrzona tą adnotacją będzie przechowywana jako wartość elementu w pliku XML (czyli to, co jest między znacznikami `<klasa>wartość</klasa>`). Tylko jedna właściwość w klasie może zawierać adnotację `@XmlValue`. Jeżeli jakaś właściwość w klasie zawiera adnotację `@XmlValue`, to wszystkie inne muszą być opatrzone adnotacją `@XmlAttribute`.

6. Adnotacja `@XmlElement` jest użyta jako domyślna dla wszystkich publicznych właściwości. Właściwość może być typu prostego lub złożonego albo listą typów prostych lub złożonych. Przyjmuje parametry:
   a. `String defaultValue` – wartość domyślna, jeżeli w pliku XML nie została wprowadzona.
   b. `String name` – nazwa znacznika elementu.
   c. `boolean required` – element jest wymagany, jeżeli `true` to w pliku XML Schema element będzie oznaczony atrybutem `minOccurs="1"` lub `minOccurs="0"` jeżeli wartość parametru wynosi `false`. Natomiast w obu przypadkach `maxOccurs="1"` dla właściwości pojedynczej, natomiast dla listy `maxOccurs="unbounded"`.

7. Adnotacja `@XmlAttribute` oznacza, że właściwość w pliku XML będzie reprezentowana jako atrybut elementu reprezentowanego przez klasę. Właściwość taka musi być typu prostego, nie może być też listą. Przyjmuje parametry:
   a. `String name` – nazwa atrybutu w XML.
   b. `boolean required` – atrybut jest wymagany, w XML Schema będzie opatrzony atrybutem `use="required"`.

8. Klasa której element będzie korzeniem pliku XML, musi zawierać adnotację:

```java
@XmlRootElement
```

Adnotacja ta przyjmuje parametr `String name`, który określa nazwę znacznika, pod którym będzie reprezentowany obiekt tej klasy, np.:

```java
@XmlRootElement(name="books")
```

Domyślnie parametr `name` przyjmuje wartość taką, jak nazwa klasy z przekonwertowaniem pierwszego ciągu wielkich liter na małe.

Nie może być dwóch elementów o takiej samej nazwie. W szczególności należy uważać na taką konstrukcję:

```java
@XmlElement(name="inaczej")
private String costam;

public String getCostam() {
   return costam;
}
```
public void setCostam(String costam) {
    this.costam = costam;
}

Biblioteka JAXB używa właściwości costam z bezpośrednim dostępem do niej, a potem jeszcze raz próbuje użyć właściwości dostępnej za pośrednictwem getterów i setterów i użyje dla niego domyślnego znacznika „costam”. Wczytanie takiej klasy spowoduje nadpisanie jednej wartości drugą. W tym przypadku autor za pewne chce, by biblioteka JAXB wprowadzała dane do klasy za pomocą getterów i setterów, więc adnotacja @XmlElement powinna zostać przeniesiona przed metodę getCostam(). Natomiast jeżeli autor chce, by użytkownik używał wartości za pomocą getterów i setterów, a biblioteka JAXB miała dostęp bezpośredni, powinien przed getCostam() wstawić adnotację @XmlTransient.

Aby wygenerować plik schematu XML, musimy najpierw utworzyć kontekst JAXB:

```java
JAXBContext context = JAXBContext.newInstance(Books.class);
```


```java
final File outFile = new File("books.xsd");
    context.generateSchema(new SchemaOutputResolver() {
        @Override
        public Result createOutput(String string, String string1) throws IOException {
            StreamResult result = new StreamResult(outFile);
            result.setSystemId(outFile.toURI().toURL().toString());
            return result;
        }
    });
```

Do zapisu stanu klasy do pliku XML wykorzystuje się klasę Marshall. Obiekt tej klasy tworzy się z obiektu kontekstu JAXB w następujący sposób:

```java
Marshaller m = context.createMarshaller();
```

Ustawienie parametru Marshall.JAXB_FORMATTED_OUTPUT na true spowoduje, że wygenerowany plik XML będzie sformatowany za pomocą wcięć co uczyni go bardziej czytelnym dla człowieka. Parametr ten ustawi się za pomocą instrukcji:

```java
m.setProperty(Marshall.JAXB_FORMATTED_OUTPUT, true);
```
Skoro wygenerowaliśmy już plik books.xsd, warto go wykorzystać dopisując go w parametrze Marshaller.JAXB_NO_NAMESPACE_SCHEMA_LOCATION.
Zapis stanu obiektu klasy dokonuje się w chwili wywołania instrukcji:

```java
m.marshal(books, System.out);
```


Do tworzenia obiektu klasy ze stanu zapisanego w pliku XML służy klasa Unmarshaller. Obiekt tej klasy tworzy się z obiektu kontekstu JAXB w następujący sposób:

```java
Unmarshaller u = context.createUnmarshaller();
```

Klasy Unmarshaller dokonuje walidacji na podstawie adnotacji. Możemy ją jednak zmusić do dodatkowej walidacji na podstawie podanego przez nas pliku XML Schema. Dla pliku books.xsd możemy wykonać taką instrukcję:

```java
u.setSchema(SchemaFactory.newInstance(
    XMLConstants.W3C_XML_SCHEMA_NS_URI)
    .newSchema(new File("books.xsd")));
```

Ma to jednak sens jedynie wtedy, kiedy dokonaliśmy ręcznych modyfikacji pliku books.xsd i muszą one zostać uwzględnione.

Za utworzenie nowego obiektu ze stanu zapisanego w pliku XML odpowiada metoda unmarshal. Ma ona wiele wariantów, w tym dla obiektów klasy InputStream i File. Jeżeli stan chcemy wczytać z pliku books.xml, musimy wywołać instrukcję:

```java
Books books = (Books) u.unmarshal(xmlFile);
```

5.5. Przebieg laboratorium

Druga część instrukcji zawiera zadania do praktycznej realizacji, które demonstrują zastosowanie technik z omawianego zagadnienia.

5.5.1. Zadanie 1. Zapis pliku XML w Java

Celem zadania jest zaimplementowanie programu, który stworzy plik books.xml i zapisze w nim informacje o wprowadzonych książkach. Wygenerowany plik books.xml musi być zgodny z dokumentem schematu XML zapisanym w pliku books.xsd (patrz rozdział: Schematy XML).

Należy pamiętać, że książka musi mieć co najmniej jednego autora i tytuł w co najmniej jednym języku.
Wszystkie informacje na temat książek program ma pobierać ze standardowego wejścia według następującego algorytmu:
1. Wypisz: „Czy wprowadzić nową książkę?”.
2. Jeżeli wprowadzono „nie” zakończ algorytm i zapisz dane na dysku.
3. Wypisz: „Podaj autora książki” i pobierz odpowiedź.
4. Jeżeli wprowadzona odpowiedź nie jest linią pustą to przejdź do pkt. 5.
5. Wypisz: „Podaj język tytułu” i pobierz odpowiedź.
7. Jeżeli odpowiedzi z pkt. 5 i 6 nie są puste to przejdź do pkt. 8.

Program powinien być tak napisany, aby możliwe było podanie więcej niż jednego autora, oraz tytułu w kilku językach. Dane te powinny następnie zostać odzwierciedlone w pliku XML.

5.5.2. Zadanie 2. Odczyt pliku XML z walidacją w Java

Celem zadania jest zaimplementowanie programu, który wczyta stworzony w poprzednim zadaniu plik books.xml, dokona jego walidacji za pomocą pliku books.xsd, następnie wypisze informacje o zapisanych książkach w formacie:

<table>
<thead>
<tr>
<th>Tytuł w języku</th>
<th>Tytuł w języku</th>
<th>Autorzy</th>
<th>ISBN</th>
</tr>
</thead>
</table>

Oznaczenia w klamerkach:
- title[n] – n-ty wariant tytułu
- title[n].lang – język n-tego wariantu tytułu
- author[n] – n-ty autor
- isbn – numer ISBN

5.5.3. Zadanie 3. Biblioteka JAXB i obiektów POJO

Napisać program wykonujący identyczne zadanie co program z zadania 1, ale z wykorzystaniem biblioteki JAXB i obiektów POJO. Program musi w odpowiedni sposób implementować klasy Books (korzeń pliku XML), Book i Title (tytuł książki z informacją o języku).


6. Web service, WSDL i UDDI

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące architektury SOA. Zostały wyjaśnione niektóre pojęcia, wprowadzające użytkownika w tę tematykę. Następnie opisany został bazowy model architektury, mający na celu uświadomienie użytkownikowi zachodzących wewnątrz procesów.

6.1. Tematyka laboratorium

Tematyką laboratorium jest zapoznanie się z architekturą SOA, a konkretnie z modelem bazowym architektury, zawierającym podstawowe elementy zaliczane w skład technologii zorientowanej na usługi. Do tych elementów należą:

- usługi sieciowe, popularnie zwane serwisami webowymi (ang. Web Services),
- język opisu usługi sieciowej WSDL (ang. Web Services Description Language),
- protokół SOAP (ang. Simple Object Access Protocol),
- uniwersalny rejestr usług UDDI (ang. Universal Description, Discovery and In-tegration).

Architektura zorientowana na usługi wykorzystująca wspomniane wyżej elementy (zwłaszcza usługi sieciowe Web Services) często spotykana jest pod nazwą WSOA.

6.2. Cel laboratorium

Celem laboratorium jest poznanie budowy architektury SOA, a dokładniej WSOA, składającej się z trzech podstawowych elementów:

- Usługi klienta,
- Rejestr usług,
- Usługodawcy.

Usługi sieci – popularne Web Serwisy, stanowią podstawę SOA wcielając się zarówno w rolę usługodawcy jak i klienta. WSDL opisuje poszczególne usługi sieciowe, które z kolei można wyszukać w rejestrze UDDI. Wiadomości przesyłane pomiędzy Web Serwisami oparte są o standard SOAP.

Dzięki temu laboratorium będziesz miał świadomość, jak należy stworzyć system wykorzystujący elementy architektury zorientowanej na usługi.
6.3. Zagadnienia do przygotowania

Przed przystąpieniem do realizacji laboratorium należy zapoznać się z zagadnieniami dotyczącymi:

- umiejętność posługiwania się środowiskiem Netbeans [2]
- znajomość wykorzystania serwera Glassfish [5]
- programowania w języku Java [1, 2]
- wstępne informacje na temat SOA [3, 4, 6]

Należy także dobrze przyswoić sobie zagadnienia omawiane w niniejszym dokumencie pozwalające na dalszą pracę w tej tematyce:

- zasada działania usług webowych,
- tworzenie nagłówków SOAP,
- zasada działania języka opisującego usługi webowe,
- przydatność serwera UDDI,


6.4. Opis laboratorium

Podczas laboratorium stworzony zostanie projekt składający się z elementów tworzących podstawowy model architektury zorientowanej na usługi. Laboratorium składać się będzie z dwóch części: teoretycznej i praktycznej. W pierwszej części użytkownik posiądzie wiedzę teoretyczną na temat modelu bazowego architektury zorientowanej na usługi (ang. Services Oriented Architecture). W tej części przedstawione zostaną wszystkie części architektury wprowadzając użytkownika w tematykę SOA.

Kolejna część pozwoli sprawdzić zdolności praktycznego posługiwania się architekturą zorientowaną na usługi. Najpierw stworzona zostanie prosta usługa sieciowa, dzięki której zrealizowany zostanie jeden z trzech elementów stanowiących bazowy model architektury. W kolejnym kroku zaprezentowany zostanie sposób, w jaki należy opisać tworzoną usługę tworząc odpowiednie dokumenty WSDL. W końcowej fazie laboratorium, przedstawiony zostanie sposób, w jaki można zaimplementować serwer UDDI stanowiący kolejny element modelu bazowego. W tej części użytkownik będzie mógł się przekonać jak należy zarejestrować daną usługę na serwerze.
6.4.1. SOA wiadomości wstępne

W przypadku realizacji architektury zorientowanej na usługi nie sposób nie wspomnieć o usługach sieciowych, z którymi SOA jest często ściśle powiązana. Usługi sieciowe zwane Web Services są jednym z możliwych rozwiązań dla realizacji koncepcji SOA. Najbardziej znany standardem w technologii SOA, korzystającej z usług sieciowych jest język opisujący wspomniane usługi widniejący pod nazwą WSDL (ang. Web Service Description Language), SOAP\(^1\), Universal Description Discovery and Integration (UDDI). Rys. 1 pokazuje użycie tych standardów tworząc model bazowy architektury opartej na usługi. W związku z funkcją, jaką może pełnić ta architektura (tzn. w związku z silnym powiązaniem z Web-Serwisami) architektura ta często spotykana jest pod nazwą WSOA.

Rys. 1. Bazowy model SOA

6.4.2. Usługi sieciowe Web Services

Usługa świadczona poprzez sieć telekomunikacyjną, a w tym sieć komputerową, w szczególności przez Internet. Według organizacji W3C Web-Services to:

„System programowy przeznaczony do wspierania interakcji maszyna-maszyna w sieci. Posiada interfejs opisany w formacie zrozumiałym dla maszyn nazywany WSDL. Inne systemy współpracujące z usługą sieci Web, w sposób określony przez jego opis za pomocą komunikatów SOAP, zazwyczaj przekazywane za pomocą protokołu HTTP wykorzystując serializację XML.”

Definicja ta pozwala na lуźne powiązanie, które jest jedną z cech charakterystycznych architektury SOA. Usługi sieciowe powinny stosować standardy sieciowe oraz winny być stosownie opisane poprzez język WSDL przybliżony w kolejnym dziale. Z tego powodu serwis WWW może być udostępniony poprzez sieć internetową. Ponadto, stosowanie standardowych protokołów i opis pozwalają na lуźne wiązanie i interakcję pomiędzy dowolnymi komputerami. Web-Services umożliwiają dostęp do aplikacji lub innych usług sieciowych.

---

\(^1\) SOAP – protokół wywoływania zdalnego dostępu do obiektów oparty o język XML, do wersji 1.2 skrót ten związany był z nazwą *Simple Object Access Protocol*. 

Należy podkreślić, że serwisy internetowe nie są obowiązkowe dla SOA, jednak najczęściej architektura ta oparta jest o Web-Services – stąd nazwa WSOA.

6.4.3. WSDL – Web Services Description Language

Według organizacji W3C dokument WSDL definiuje usługi jako zbiory punktów końcowych sieci lub w portach. W WSDL abstrakcyjna definicja punktów końcowych i wiadomości jest oddzielona od ich budowy lub konkretnego powiązania formatu danych. To pozwala na ponowne abstrakcyjne definicje: wiadomości, które są abstrakcyjnym opisem przesyłanych danych oraz typy portów, które są abstrakcyjną kolekcją operacji. Konkretny protokół wraz z odpowiednim formatem danych dla danego typu portu stanowi wielokrotnie wykorzystywane powiązanie. Port jest zdefiniowany przez skojarzenie adresu sieciowego z wiązania wielokrotnego użytku, natomiast zbiór portów zdefiniuje usługę. Dlatego też dokument WSDL korzysta z następujących elementów definicji usług sieciowych:

- Tytyp – pojemnik na definicje typów danych za pomocą jakiegoś systemu typu (np. XSD).
- Wiadomość – przesyłane dane.
- Operacje – skrócony opis funkcjonalności wykonywanej przez usługę.
- Typy portów – zbiór operacji obsługiwanych przez jeden lub więcej punktów końcowych.
- Powiązania – konkretny protokół i specyfikacja formatu danych dla konkretnego typu portu.
- Port – pojedynczy punkt końcowy zdefiniowany jako połączenie wiązania i adresu sieciowego.
- Usługa – zbiór powiązanych punktów końcowych.

Ważne jest, aby zauważyć, że WSDL nie wprowadza nowego języka definicji typu. WSDL dostrzega potrzebę rozszerzonego opisu formatów wiadomości oraz wsparcia schematów opisanych poprzez XML. Jednakże, ponieważ nie można się spodziewać jednego typu stosowanej gramatyki do opisu wszystkich formatów wiadomości obecnych i przyszłych, WSDL pozwala na użycie innych języków definicji typu za pośrednictwem rozszerzeń.

Oprócz podstawowych ram definicji usługi, specyfikacja ta wprowadza specyficzne wiązanie rozszerzeń dla następujących protokołów i formatów komunikatów:

- SOAP 1.1,
- HTTP GET / POST,
- MIME.

6.4.4. Praktyka – Web Services oraz WSDL

W tej części stworzony zostanie przykładowy Web-Services umożliwiający dodawanie dwóch liczb. Do serwisu wygenerowany zostanie dokument WSDL, analizowana treść dokumentu powinna pomóc w zrozumieniu zasady działania WSDL.
Modele programowania

Podczas implementacji usług sieciowych nazywanych Web-Services istnieje kilka modeli programowania danej usługi. Powszechnie znane są dwa modele, które są wspierane przez środowisko NetBeans, w którym powstanie przykładowa usługa.

- **REST-based** – REpresentational State Transfer, to nowy sposób tworzenia i komunikacji z usługami sieciowymi. W REST, źródła mają URI i są manipulowane poprzez operacje w nagłówkach HTTP.
- **SOAP/WSDL-based** – w tradycyjnych modelach usług sieciowych, interfejs Web-Serwis uwidaczniany jest za pomocą dokumentów WSDL, mających strukturę pliku XML. Przesyłana wiadomość jest w SOAP, który również opisany jest za pomocą struktury XML. W naszym projekcie wykorzystana zostanie ta metoda programowania.

Zakres działań, które mogą być przekazywane w SOAP jest znacznie szerszy niż to, co jest dostępne w REST, zwłaszcza w zakresie bezpieczeństwa. W SOAP usługi nadają się do cięższych aplikacji z wykorzystaniem skomplikowanych operacji, zapewniają zaawansowane zabezpieczenia, niezawodność lub inne standardy WS-*.


Implementacja prostej usługi


W kolejnych krokach należy wybrać odpowiednią charakterystyczną nazwę dla projektu oraz określić serwer, w jakim będzie pracował. W naszym projekcie wykorzystany będzie serwer GlassFish instalowany w pakietie z GlassFish Open ESB w wersji 2.1. Istnieje możliwość doinstalowania serwera w przypadku braku zainstalowanego za pomocą przycisku Add.

Następnie za pomocą prawego przycisku myszy klikamy na projekcie i wybieramy New → Web Service, po czym w oknie odpowiednio nazywamy stworzoną usługę sieciową.

Wygenerowany zostanie pusty dokument, do którego należy zdefiniować jakąkolwiek usługę, by projekt mógł zostać skompilowany poprawnie. W celu zdefiniowania usługi, można posłużyć się kreatorem, bądź zbudować wszystko
ręcznie. Za pomocą kreatora w wygenerowanym pliku, należy zaznaczyć pozycję Design, a następnie wprowadzić konkretną operację. W naszym projekcie będzie to operacja dodawania.

W wyniku powyższych operacji uzupełniona zostanie treść strony występującej po kliknięciu przycisku Add w tworzonym Web Services. W pustej metodzie należy dokonać stosownych zmian przedstawionych poniżej.

```java
package Calculator;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

@WebService()
public class CalcWS {

    @WebMethod(operationName = "Add")
    public Integer Add(@WebParam(name = "first") int first, @WebParam(name = "second") int second) {
        int result = first + second;
        return result;
    }
}
```

Następnie należy skompilować cały projekt oraz opublikować go na serwerze GlassFish za pomocą polecenia Deploy. Wtedy zostanie stworzony Web Services na serwerze, dzięki czemu będzie można podglądać opisujący go dokument WSDL. W celu sprawdzenia poprawności opublikowania usługi należy załogować się poprzez konsole administracyjną do serwera, po czym sprawdzić dostępność usługi. Informacje o tym jak można to zrobić, znajdują się w literaturze.

**Test WebSerwis CalcWS**

W tym punkcie opisane będą testy sprawdzające poprawność działania usługi. Po wygenerowaniu usługi CalcWS zgodnie z powyższą instrukcją należy prawym przyciskiem myszy kliknąć na stworzoną usługę (w folderze Web Services projektu), a następnie wybrać Test Web Service. Wtedy sprawdzony zostanie wygenerowany automatycznie dokument WSDL poprzez serwer GlassFish, a następnie wygenerowana zostanie strona zawierająca test operacji dodawania, jak poniżej:
CalcWSService Web Service Tester

This form will allow you to test your web service implementation. (WSDL File)

To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

```java
public abstract int org.me.calculator.CalcWS.opAdd(int,int)
```

Rys. 2. Wygenerowana strona testów

Dokument WSDL usługi


Rys. 3. Dokument WSDL

6.4.5. Universal Description Discovery and Integration

Universal Description Discovery and Integration (UDDI) to otwarty standard, sponsorowany przez OASIS. Celem UDDI jest dostarczanie informacji o usługach, które mogą zostać udostępnione. UDDI działa jako usługa katalogowa, która zawiera publikacje usług. Chodzi o to, że usługi mogą dynamicznie znaleźć inne usługi bez interwencji człowieka. Aby to osiągnąć, UDDI opiera się głównie na dokumencie WDSL opublikowanym przez usługodawcę. W praktyce serwery UDDI są potencjalnymi źródłami ataków, w związku z czym są one trudne do utrzymania.

Wpis katalogu UDDI składa się z trzech części. „Białe strony” zawierają opis firmy, która oferuje usługi (nazwa, adres, dane kontaktowe itp.). „Żółte strony” zawierają kategorie branżowe według standardowych taksonomii, takich jak NAICS (system klasyfikacji branż Ameryki Północnej) lub Międzynarodowa Standardowa Klasyfikacja Rodzajów Działalności. „Strony zielone” zawierają na tyle szczegółowy opis interfejsu programistycznego usługi, że programista może na jego podstawie napisać aplikację wykorzystującą daną usługę XML Web Service. Usługi są definiowane za pomocą dokumentów UDDI o nazwie Type Model lub tModel. W wielu przypadkach tModel zawiera plik WSDL opisujący interfejs SOAP usługi XML Web Service, ale dokumenty tModel są na tyle elastyczne, że mogą opisać usługi dowolnego typu.

W katalogach UDDI przewidziano różne sposoby wyszukiwania usług potrzebnych do zbudowania aplikacji. Na przykład można szukać dostawców usług z konkretnego regionu geograficznego lub firm określonego typu. Odpowiedzią katalogu UDDI na zapytanie są informacje, dane kontaktowe, odnośniki i dane techniczne, umożliwiające wybranie usługi spełniającej nasze oczekiwania.


**Praktyka – wprowadzenie**


**Instalacja serwera**

Serwer UDDI, który wybrano w poprzednim punkcie może być zainstalowany na platformie Windows bądź Linux. W kolejnych krokach zaprezentowany będzie sposób instalacji serwera na platformie Windows. Projekt serwera pobieramy i rozpakowujemy w konkretne miejsce na dysku.
Uruchomienie serwera

Zanim serwer zostanie uruchomiony, należy dokonać drobnych zmian konfiguracyjnych. Po pierwsze należy upewnić się, że zmienne systemowe są odpowiednio ustawione, np. zgodnie z poniższymi informacjami:

JAVA_HOME={JavaHOME}\jdk1.6.0_18  
JRE_HOME={JavaHOME}\jre6  
### {JavaHOME} to miejsce instalacji platform Java, np.  
C:\Program Files\Java

Dobrą praktyką jest również zmiana portu, zamiast domyślnego 8080. Port ten jest często wykorzystywany przez różne systemy / serwery instalowane na lokalnej maszynie. W celu zmiany portu należy edytować poniższe pliki, zmieniając wartość 8080 na inną, np. 8095:

{JUDDI_PATH}\conf\server.xml  
{JUDDI_PATH}\webapps\juddiv3\WEB-INF\classes\juddiv3.properties  
{JUDDI_PATH}\webapps\uddi-portlets\WEB-INF\classes\juddiv3.properties  
{JUDDI_PATH}\webapps\uddi-portlets\WEB-INF\classes\META-INF\uddi.xml  
### {JUDDI_PATH} to miejsce rozpakowania / instalacji projektu

Ostatnią rzeczą, którą można skonfigurować przed uruchomieniem serwera to nadanie odpowiednich uprawnień poszczególnym użytkownikom. W tym celu należy edytować poniższy plik:

{JUDDI_PATH}\conf\tomcat-users.xml  
<?xml version='1.0' encoding='utf-8'?>  
  <tomcat-users>  
    <role rolename="tomcat"/>  
    <role rolename="manager"/>  
    <user username="root" password="p@wel" roles="tomcat,manager"/>  
    <user username="uddi" password=" p@wel " roles="tomcat,manager"/>  
    <user username="pawel" password="p@wel" roles="tomcat,manager"/>  
  </tomcat-users>  
### {JUDDI_PATH} to miejsce rozpakowania / instalacji projektu

W celu uruchomienia serwera wystarczy przejść do katalogu instalacji projektu JUDDI, a następnie odnalezieniu skryptów startup.bat (uruchomienie serwera) i shutdown.bat (wyłączenie). Należy jednak pamiętać, aby poprawnie ustawić zmienne środowiskowe wymienione w pierwszym kroku.
Weryfikacja instalacji

Rys. 4. Weryfikacja instalacji serwera (http://localhost:8095/juddiv3/)

Zarządzanie usługami

Serwer JUDDI posiada klienta, który może łączyć się z rejestrem poprzez udostępnione przez niego usługi typu Web-Serwis. W związku z tym poprzez klienta, możemy zarządzać publikacją czy rejestracją usług w serwerze.

W tym celu należy stworzyć nowy projekt w środowisku np. NetBeans, w którym należy dołączyć:

juddi-client-3.0.4.jar
commons-collections-3.2.1.jar
commons-configuration-1.5.jar
commons-lang-2.3.jar
commons-logging-1.1.jar
log4j-1.2.13.jar
uddi-ws-3.0.4.jar

Następnie należy w projekcie stworzyć pliki uddi.xml oraz log4j.properties, które powinny znajdować się w katalogach kolejno META-INF oraz <default package>.

Plik log4j.properties zawiera parametry programu logującego, natomiast plik uddi.xml zawiera adres serwera oraz port, który należy poprawnie ustawić.
Implementacja usług

Zgodnie z rozpiską poprzednich rozdziałach serwer zawiera możliwość publikacji i rejestracji usług. W tym celu dostarczone są pewne metody, które należy używać zgodnie ze schematem, który prezentuje poniższy rysunek.

![Diagram: UDDI Implementation Hierarchy](image)

Rys. 5. Hierarchia implementacji UDDI

6.5. Przebieg laboratorium

Druga część instrukcji zawiera zadania do praktycznej realizacji, które demonstrują zastosowanie technik z omawianego zagadnienia.

6.5.1. Zadanie 1. Implementacja usługi CalcWS

Zaimplementować aplikację zawierającą serwis CalcWS, a następnie rozbudować ją o kolejne moduły. Dodatkowo należy zaimplementować stronę główną, z której będzie można uruchomić usługę. W celu realizacji zadania należy wykonać następujące kroki:

1. Zaimplementować program zgodnie ze wskazówkami w dokumencie.
2. Dodać kolejne operacje dostępne w usłudze:
   a. odejmowanie,
   b. mnożenie,
   c. dzielenie.
3. Stworzyć stronę html:
   a. wywołać usługę ze strony głównej,
   b. umożliwić wprowadzanie danych,
   c. wybór odpowiedniej operacji.
5. Przeanalizować dokument WSDL przed dodaniem operacji jak i po dodaniu.
6.5.2. Zadanie 2. Stworzenie klienta

Zadanie polega na stworzeniu klienta do usługi zaimplementowanej w poprzednim ćwiczeniu. W tym przypadku należy stworzyć nowy projekt, analogicznie jak podczas realizacji części teoretycznej laboratorium. Następnie z poziomu klienta, wywołać stworzoną usługę i zaprezentować w przejrzystej formatce wyniki.

W celu realizacji zadania należy zrealizować następujące kroki:

1. Opublikować usługę z zadania 1, rozszerzoną o operacje:
   a. mnożenia,
   b. dzielenia,
   c. odejmowania.
2. Stworzyć aplikację klienta:
   a. typ aplikacji Java EE,
   b. zaimportować usługę za pomocą komendy Web Service klient:
      i. należy kliknąć prawym przyciskiem myszy na New → Web Service Klient,
      ii. za pomocą dokumentu WSDL zaimportować projekt.
3. Stworzyć przejrzysty interfejs graficzny.


Celem zadania jest zaimplementowanie serwera rejestrów UDDI. Można wykorzystać dowolny serwer, zaleca się aby był to serwer zaprezentowany w dokumencie. Na serwerze należy wdrożyć usługę stworzoną w zadaniu nr 1 o nazwie CalcWS. Dodatkowo należy stworzyć klienta, który będzie umożliwiał przeprowadzenie operacji związanych z zarządzaniem usługami w tych ich publikowaniem i wyszukiwaniem.
Zadanie to ma na celu uświadomienie, jak działa serwer rejestrów występujący w bazowym modelu SOA. Dzięki temu użytkownik będzie mógł sprawdzić, w jaki sposób usługa zostanie zarejestrowana bądź w jaki sposób wyszukiwać usługi na serwerach UDDI.

W celu realizacji zadania należy zrealizować następujące kroki:

1. Pobrać serwer rejestrów
   a. JUDDI,
   b. dowolny inny uzasadniając wybór.

2. Przeprowadzić proces instalacji:
   a. Zmiana portu domyślnego na 8095.

3. Skonfigurować listy dostępowe:
   a. stworzenie polityk bezpieczeństwa,
   b. ustawienie uprawnień użytkowników,
   c. tworzenie użytkowników.

4. Implementacja klienta komunikującego się z rejestrem UDDI:
   a. można wykorzystać biblioteki do uruchomienia klienta dostarczonego do projektu.

5. Funkcjonalności klienta:
   a. rejestracja zadanej usługi,
   b. wyrejestrowanie zadanej usługi,
   c. przeszukiwanie czy zadana usługa istnieje,
   d. wyciąganie informacji dodatkowych na temat usługi na podstawie dokumentu WSDL,
   e. w przypadku korzystania z wbudowanego klienta, również należy zrealizować wszystkie wymienione punkty.
7. Szyna ESB

Instrukcja składa się z dwóch części. W pierwszej części zostanie przedstawiony OpenESB i jego rys technologiczny. Następnie w kolejnej części zostanie przedstawiony przykład zastosowania OpenESB na konkretnym problemie, aby ukazać jaka siła tkwi w ESB i dlaczego ta koncepcja powinna być szerzej stosowna.

7.1. Tematyka laboratorium

Tematyką tej instrukcji jest programowanie aplikacji klient-serwer w oparciu o technologię SOA (Service Oriented Architecture) przy zastosowaniu koncepcji użycia szyby komunikacyjnej ESB (Enterprise Service Bus). Jako przykładowa szyna zostanie użyty serwer OpenESB. Aby w pełni zrozumieć jak on działa zostanie wpierw przedstawiona jego architektura i moduły, z których się składa.

7.2. Zagadnienia do przygotowania

Do wykonania zadań opisanych w niniejszym laboratorium niezbędne jest pobranie oraz instalacja pakietu oprogramowania OpenESB spod adresu: http://hudson.openesb-dev.org:8080/hudson/job/openesb-installers-legacy/lastSuccessfulBuild/artifact/dist/

Przed przystąpieniem do realizacji laboratorium należy zapoznać się z zagadnieniami dotyczącymi:
- Jak działają systemy rozproszone.
- Jak działa model klient – serwer.
- Ogólna znajomość technologii SOA.
- Znajomość jak działają Web Services.
- Ogólna znajomość koncepcji szyny ESB.
- Znajomość NetBeans.


7.3. Opis laboratorium

ESB (ang. Enterprise Service Bus) jest korporacyjną szyną usług oraz warstwą pośredniczącą w wymianie danych.

Aby lepiej wyjaśnić zalety szyny ESB, zostanie najpierw przedstawiona klasyczna architektura systemów opartych na paradygmacie SOA bez szyny ESB.
7.3.1. Architektura klasyczna

Klasyczną architekturę przedstawia Rys. 2. Architektura klasyczna. U góry znajdują się elementy wsparcia biznesowego dla klientów, sprzedawców oraz marketingowców. Na dole znajdują się przykładowe systemy świadczące usługi dla nich takie jak systemy rachunkowe, fakturujące, reklamacyjne oraz sprawdzające wiarygodność kart kredytowych. Jest to wielokrotnie używany i powielany schemat, ale niestety rozwiązanie jest obarczone wieloma niedogodnościami oraz wywodzącymi się z wad nieprawidłowościami. Jak można szybko dostrzec znajduje się tam wiele połączeń pomiędzy systemami, co skutkuje problemami podczas zmian w systemach i ich czasochłonnością. Każdy z systemów zawsze musi dokładnie wiedzieć, gdzie znajdują się usługi, z których chce w danej chwili skorzystać. W sytuacji gdy któryś z systemów zmieni swoją lokalizację wszystkie inne systemy muszą być o tym poinformowane i odpowiednio skonfigurowane. Jeszcze większą komplikację może powodować zmiana funkcjonalności któregoś z systemów, które są używane z zewnątrz. Problemem jest również dostrojenie komunikacji pomiędzy serwisami, kiedy pochodzą one od innych dostawców oraz każdy z nich komunikuje się w inny sposób. Zatem skupiamy się w głównej mierze na zapewnianiu kompatybilności, a nie na wykorzystaniu w pełni możliwości systemów. Problemem największym może być w tej sytuacji konieczność stworzenia nowych systemów lub integracji istniejących z kolejnymi systemami. Stopień komplikacji takiego rozwiązania będzie stale rósł, powodując coraz to trudniejsze utrzymanie i rozwój poszczególnych systemów. W takiej sytuacji przychodzi z pomocą szyna ESB.
7.3.2. Architektura ESB

Przykład połączonych serwisów z szyną komunikacji ESB obrazuje Rys. 3. Od razu rzuca się w oczy zredukowana ilość połączeń. Każdy z systemów i serwisów musi być podłączony do szyny ESB i w każdym przypadku tylko z nią się komunikować. Takie podejście rozwiązuje wiele wad wcześniej omawianej przykładowej klasycznej koncepcji. Z racji, że ilość połączeń jest mniejsza, dodawanie nowych serwisów polega tylko na połączeniu ich z szyną ESB. W tym przypadku nie trzeba przejmować się również z punktu widzenia konsumenta niezgodnością protokołów ponieważ ich konwersja odbywa się automatycznie na szynie ESB. W takim przypadku nie interesują nas kwestie jakimi protokołami wystawiane są usługi przez serwisy. Jest to bardzo eleganckie i dużo łatwiejsze rozwiązanie z punktu widzenia rozwijającego się serwisu i systemów. Jedyną dodatkową pracą jest zarządzanie szyną ESB.

Ponadto szyna ESB jako warstwa pośrednicząca w wymianie danych pozwala na:

- monitorowanie komunikacji,
- priorytetyzacje, filtrowanie, zabezpieczenie wywołań oraz transmisji (szfrowanie),
- skalowalność, load-balancing, etc.

7.3.3. JBI

ESB jest szyną usług, natomiast standard JBI (ang. Java Business Integration), na którym oparty jest OpenESB, definiuje standard budowy systemów integracyjnych w języku Java w oparciu o Web Services i język XML.

Komponenty dzielą się na dwie grupy:

- Binding Components (BC),
- Service Engines (SE).

**Binding Components** odpowiadają za obsługę różnych protokołów do komunikacji ze światem zewnętrznym. Podstawowymi zadaniami komponentów BC są:

- komunikacja ze światem zewnętrznym przez protokoły HTTP, JMS, RMI itp.
- normalizacja i denormalizacja otrzymanych wiadomości.

Jako przykładowe **Binding Components** w OpenESB można wymienić:

- sun-database-binding
- sun-email-binding
- sun-file-binding
- sun-http-binding
- sun-ftp-binding
- sun-jms-binding

**Service Engines** odpowiedzialne są za logikę w obrębie JBI. Nie mają możliwości komunikacji ze światem zewnętrznym, jedynie z NMR (*Normalized Message Router*). Jeżeli jakiś komponent SE potrzebuje do realizacji swojej logiki jakiejś informacji ze świata zewnętrznego, musi wysłać komunikat do komponentu BC, ale tylko i wyłącznie poprzez NMR. Logika komponentu SE nie jest logiką typowo aplikacyjną, a jedynie taką, która służyć ma konwersji wiadomości bądź tworzeniu zasad routingu itp. W związku z czym należy szczególnie uważać na logikę, która implementują. Dobrymi przykładami SE mogą być: silnik reguł, silniki BPEL, transformacja XSLT czy na przykład różnego rodzaju skrypty.

W OpenESB przykładowymi komponentami **Service Engines** można wymienić:

- sun-bpel-engine
- sun-edm-engine
- sun-javaee-engine
- sun-pojo-engine
- sun-xslt-engine

### 7.3.4. OpenESB

Jest to opensource’owa szyna ESB, która wspiera implementację SOA. OpenESB działa na serwerze GlassFish. Cały silnik OpenESB opiera się na opisanym wyżej JBI, które umożliwia serwisom integrację pomiędzy sobą z zachowaniem luźnych powiązań, niezależnie od wytwórcy oprogramowania. Przypomina to swego rodzaju standard Plug & Play dla systemów świadczących usługi i systemów, które korzystają z dostępnych usług. Na Rys. 4 został przedstawiony uproszczony model budowy szyny OpenESB, korzystający z JBI.

OpenESB zawiera kilka gotowych komponentów, które w większości zastosowań w zupełności wystarczą:
- `sun-javaee-engine` jest to komponent SE, którego zadaniem jest bezpośrednia komunikacja z aplikacjami JavaEE udostępniającymi swoje usługi przez interfejs Web Services. Bez tego komponentu komunikacja z tymi aplikacjami musiała by się odbywać za pomocą któregoś z komponentów BC.
- `sun-bpel-engine` to komponent SE, któremu poświęcona jest dalsza część instrukcji oraz następne ćwiczenie. Jego zadaniem jest nadzór nad wykonaniem procesów biznesowych zapisanych w języku BPEL.
- `sun-http-binding` to komponent BC, umożliwiający komunikację za pomocą wiadomości SOAP przesyłanych za pomocą HTTP.
- `sun-ftp-binding` to komponent obsługujący zapis i odczyt na serwerach FTP. Zapewnia między innymi cykliczne odpytywanie serwera. Podobnie działa komponent `sun-file-binding`, tylko że odbywa się to za pomocą URI.
7.3.5. NetBeans i OpenESB

Serwer aplikacji OpenESB jest rozpowszechniany wraz ze środowiskiem programistycznym NetBeans wzbogaconym o odpowiednie wtyczki, które dostarczają graficznych narzędzi do rozwoju aplikacji działających z wykorzystaniem środowiska JBI.

Wtyczki te ułatwiają administrację serwerem OpenESB, edycję plików WSDL, XSD, BPEL oraz Composite Applications.

Na Rys. 5 przedstawiono rozwinięty panel serwera GlassFish z zainstalowanym OpenESB. Z tego panelu możliwe jest uruchamianie oraz zatrzymywanie, instalacja i deinstalacja oraz konfiguracja komponentów BC i SE. Ponadto umożliwia instalację/deinstalację bibliotek współdzielonych (Shared Libraries) oraz wdrożenie pakietów usługowych (Service Assemblies).

Biblioteki współdzielone są zwykłymi bibliotekami, które udostępniają swoje klasy innym komponentom. Pakiety usługowe są to paczki zawierające w sobie nasze aplikacje oraz odpowiednią konfigurację komponentów BC, która zostanie użyta podczas komunikacji naszej aplikacji z komponentem BC.

Rys. 5. Administracja komponentami JBI z poziomu NetBeans

7.3.6. BPEL Hello World

Wejściem tej funkcji jest parametr name, a wartością zwracaną ma być ciąg znaków w postaci:

```
Hello: {name}
```

Gdzie w miejsce {name} zostanie wstawiona zawartość parametru name.

W środowisku NetBeans tworzymy nowy projekt z szablonu BPEL Module (znajduje się on w kategorii SOA). Nadajemy mu nazwę HelloBPEL. Został stworzony nowy, pusty projekt z plikiem helloBPEL.bpmel. Ten plik odkładamy na chwilę na bok ponieważ musimy najpierw stworzyć odpowiedni plik WSDL, który opisuje interfejs naszej aplikacji.

Klikamy prawym przyciskiem na pień drzewa projektu HelloBPEL i wybieramy New → WSDL Document. W oknie File Name wprowadzamy wartość „helloBPEL”. Następnie w WSDL type wybieramy „Concrete WSDL Document”. Jako Binding ustawiamy „SOAP”, a jako Type ustawić wartość „RCP Literal”.

Na następnej zakładce określamy wejście i wyjście. Na wejściu chcemy tylko jeden parametr typu string o nazwie name więc polu Input definiujemy nowy Message Part Name o nazwie name, natomiast w kolumnie Element Or Type ustawić xsd:string. Na wyjściu ma pojawić się również xsd:string (konfigurowany analogicznie) o nazwie response.

Następnie klikamy Next i Finish. Tak stworzony plik WSDL jest gotowy do użycia w skrypcie BPEL.

Przechodzimy do pliku helloBPEL.bpmel. Otwieramy go w edytorze ustawiając widok Design.

W pierwszej kolejności musimy utworzyć PartnerLink – obiekt, który za pomocą skrypt BPEL komunikuje się ze światem zewnętrznym. PartnerLink tworzymy z pliku WSDL przeciągając plik helloBPEL.wsdl na kółeczko znajdujące się w środku lewego, szarego marginesu graficznego edytora BPEL.

Następną rzeczą jakiej potrzebujemy, jest pobranie wiadomości z PartnerLink.

W tym celu z palety wybieramy akcję Receive i przeciągamy ją pod kółeczko znajdujące się w środkowej części graficznego edytora, pod etykietką Process Start.

Akcja Receive wymaga pewnej konfiguracji. W tym celu klikamy na niego dwukrotnie.

W kolejnym oknie są dwie bardzo ważne rzeczy jakie musimy zrobić. Pierwsza to powiązanie akcji Receive z odpowiednią operacją z PartnerLink. Druga to powiązanie akcji Receive ze zmiennej, w której zostaną umieszczone dane wejściowe.

W tym celu rozwijamy listę Partner Link i wybieramy z niej PartnerLink1, który wcześniej utworzyliśmy. Ma on tylko jedną operację, która się powinna automatycznie ustawić w polu Operation.

Następnie, aby możliwe było odwołanie się do wartości przekazanej na wejściu musimy powiązać akcję Receive ze zmiennej, do której wartość wejścia zostanie zapisana. Jeżeli taką zmiennej już posiadamy to przy Input Varible klikamy na Browse... i wybieramy odpowiednią zmiennej. Jednak w nowoutworzonym skrypcie trudno by się ona już tam znalazła więc musimy ją najpierw utworzyć klikając na przycisk Create....

Mamy już utworzoną akcję pobierającą dane z wejścia, teraz musimy utworzyć akcję, która zwraca wynik. W tym celu z palety wybieramy akcję Reply
i przeciągamy ją na kółeczko znajdujące się pod stworzoną wcześniej akcją Receive. Podobnie jak akcja Receive, akcja Reply również wymaga powiązania z odpowiednim Partner Link i zmienną, która w chwili wykonania akcji Reply przechowuje już wartość, która ma zostać zwrócona.


Rys. 6. Graficzny edytor powiązań


Wracamy do widoku Design edytora i nasz skrypt powinien wyglądać w nim jak na rysunku Rys. 7.
Moduł BPEL nie może działać samodzielnie, dlatego konieczne jest stworzenie projektu Composite Application, który łączy w sobie aplikacje pochodzące z różnych źródeł. W naszym przypadku będzie to tylko jedna aplikacja – stworzony wcześniej moduł BPEL.

Aby utworzyć Composite Application należy w środowisku NetBeans wybrać File ➔ New Project ➔ SOA ➔ Composite Application. Nazwijmy go „HelloCA”.

Następnie musimy powiązać skrypt w języku BPEL z odpowiednimi komponentami BC. Dokonuje się tego w pliku Service Assembly. Wystarczy, że w oknie edytora przeciągnijmy projekt HelloBPEL do środkowej kolumny i klikniemy Build Project na pasku narzędzi. Odpowiednia konfiguracja zostanie wykonana automatycznie i graficzny edytor Service Assembly powinien teraz wyglądać jak na Rys. 8.


Gdy w drzewie projektu rozwiniemy nasz nowoutworzony Test Case, zobaczymy plik Input i plik Output. Input zawiera wygenerowany przez kreator szablon zapytania SOAP. W naszym przypadku plik będzie miał taką zawartość:

```xml
<soapenv:Envelope
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
  http://schemas.xmlsoap.org/soap/envelope/
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:hel="http://j2ee.netbeans.org/wsdl/HelloBPEL/helloBPEL">
  <soapenv:Body>
    <hel:helloBPELOperation>
      <name>?string?</name>
    </hel:helloBPELOperation>
  </soapenv:Body>
</soapenv:Envelope>
```

Należy go otworzyć i wypełnić odpowiednio miejsca pomiędzy pytajnikami (zamiast „?string?” np. na „kowalski”). Output to wzorcowa odpowiedź, z którą będzie porównywana odpowiedź aplikacji. Nie będziemy jej wypełniać, pozostawimy ją pustą.

Test Case możemy uruchomić na dwa sposoby. Albo klikając na jeden Test Case i z menu podręcznego wybierając Run. Zostanie wtedy uruchomiony pojedynczy Test Case. Innym sposobem jest kliknąć na projekt i z menu podręcznego wybrać Test. Zostaną wtedy uruchomione wszystkie Test Case po kolei.

Rys. 9. Komunikat po uruchomieniu Test Case gdy plik wzorca odpowiedzi jest pusty

Ponieważ plik Output mamy pusty, NetBeans wyświetli komunikat taki jak na Rys. 9. Gdy klikniemy Yes to plik Output zostanie zastąpiony tym, co przyszło jako odpowiedź od aplikacji na zadane zapytanie SOAP z pliku Input. Jeżeli w miejsce ?string? wpisaliśmy „kowalski” to plik Output powinien zawierać:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
  http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <m:helloBPELOperationResponse
      xmlns:m="http://j2ee.netbeans.org/wsdl/HelloBPEL/helloBPEL">
      <response>Hello: Kowalski</response>
    </m:helloBPELOperationResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
7.3.7. Klient aplikacji HelloCA


W naszym przypadku pełny adres Web Service aplikacji HelloCA ma postać:

http://localhost:9080/helloBPELService/helloBPELPort?wsdl

Taki adres należy podać podając jako źródło pliku WSDL. Adres ten możemy odczytać z Test Case klikając prawym przyciskiem na konkretny Test Case i wybierając z menu podręcznego Properties. W polu Destination znajduje się adres URL usługi.

7.4. Przebieg laboratoryum

Drugą część instrukcji zawiera zadania do praktycznej realizacji, które demonstrują zastosowanie technik z omawianego zagadnienia.

7.5. Zadanie 1. Testowanie komponentu sun-file-binding

Zadanie polega na przerobieniu programu HelloBPEL tak, aby informacje o wprowadzanych imionach dopisywał do pliku c:\imiona.txt. W tym celu należy:

1. Utworzyć nowy plik WSDL o nazwie namesLogger.wsdl. W kreatorze tworzenia pliku należy ustawić:
   a. WSDL Type: Concrete WSDL Document
   b. Binding: FILE
   c. Type: Write
   d. File name: imiona.txt
   e. Path: C:/
   f. File Exists: Append to Existing File
2. Po otwarciu pliku helloBPEL.bpel utworzyć Partner Link do namesLogger.wsdl ale nie z lewej, lecz z prawej strony i nadać mu nazwę plLogger.
3. Z palety wybrać akcję Invoke i wstawić ją między akcję Assign a Reply.
4. Powiązać akcję Invoke z plLogger i odpowiednią nowutworzoną zmienną.
5. Kliknąć dwukrotnie na akcję Assign i przeciagnąć HelloBPELOperationIn do WriteIn→part1.
7.6. Zadanie 2. Testowanie komponentu sun-mail-binding

Zadanie polega na napisaniu aplikacji, która będzie z katalogu \texttt{c:\input} wczytywać pliki tekstowe i wysyłać je pocztą email.

W tym celu należy:

1. Utworzyć nowy projekt BPEL, a w nim dwa pliki WSDL. Pierwszy z nich o ustawieniach:
   a. WSDL Type: Concrete WSDL Document
   b. Binding: FILE
   c. Type: Pool
   d. File name: .*\.txt
   e. Zaznaczyć Is Regex
   f. Polling directory: \texttt{C:/input}

2. Natomiast drugi plik WSDL o ustawieniach:
   a. WSDL Type: Concrete WSDL Document
   b. Binding: EMAIL
   c. Type: Send Mail (using SMTP)
   d. W pola SMTP Server, Port, User Name, Password i Use SSL wprowadzić odpowiednie dane dla naszego serwera SMTP. Następnie kliknąć na Test Credentials w celu przetestowania połączenia z serwerem SMTP.

3. W pliku BPEL użyć pierwszego pliku WSDL jako lewy Partner Link, a drugiego jako prawy i dokonać odpowiednich powiązań analogicznie jak w poprzednich zadaniach.

Uwaga: pliki z \texttt{c:\input}, które pasują do wzorca (czyli wszystkie pliki .txt) są po wczytaniu usuwane z tego katalogu.

Nie tworzymy test case dla tego przykładu, ponieważ wejściem nie jest zapytanie SOAP, lecz plik. Testowanie aplikacji ma się odbyć przez skopiowanie do katalogu \texttt{c:\input} kilku plików tekstowych posiadających rozszerzenie .txt i sprawdzenie czy dotarły na skrzynkę poczty e-mail.
8. Język BPEL

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące języka BPEL oraz jego wykorzystanie w środowisku NetBeans pod kontrolą serwera aplikacji GlassFish z zainstalowanym OpenESB. Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej części laboratorium.

8.1. Tematyka laboratorium

Tematyką laboratorium jest kompozycja usług Web Services stworzonych na poprzednich zajęciach w proces biznesowy za pomocą języka BPEL. 


Język BPEL do działania potrzebuje odpowiedniego serwera aplikacji, który implementuje silnik języka BPEL. Przykładowymi darmowymi silnikami BPEL są:

- GlassFish ESB,
- Apache ServiceMix,
- Mule.

Natomiast komercyjne rozwiązania to m.in.:

- Oracle BPEL Process Manager,
- IBM WebSphere Process Server,
- Microsoft BizTalk Server.

W większości nowoczesnych silników BPEL, język ten jest zintegrowany z szyną ESB.

8.2. Cel laboratorium

Celem laboratorium jest przedstawienie języka BPEL na przykładzie jego implementacji w OpenESB, koncepcji szyny ESB oraz podanie przykładu takiej szyny bazując na OpenESB. OpenESB jest prostym, przystępnym i łatwym w użyciu rozwiązaniem. Kluczową własnością jest to, że jest darmowy. W laboratorium wykorzystana zostanie zaledwie niewielka część z wielu komponentów, które są dostępne w OpenESB. W zależności od potrzeb można dobrać gotowe komponenty i odpowiednio je skonfigurować tak, aby spełniały nasze wymagania.
8.3. Zagadnienia do przygotowania

Przed przystąpieniem do realizacji laboratorium należy zapoznać się z zagadnieniami dotyczącymi:

- protokołu SOAP,
- protokołu HTTP,
- języka XML,
- języka Java, w szczególności:
  - tworzenie aplikacji webowych i usług WebServices,
  - szyny ESB,
  - tworzenia aplikacji BPEL module i Composite Application w środowisku NetBeans

Należy także zapoznać się z zagadnieniami dotyczącymi języka BPEL:

- zagadnienie aranżacji i choreografii usług,
- typy akcji prostych w języku BPEL.


8.4. Opis laboratorium

Pierwsza wersja języka BPEL została przedstawiona w 2002 roku, a w 2003 roku została przejęta przez OASIS (Organization for the Advanced of Structured Information Standards). Obecna wersja 2.0 ukazała się w 2007 roku.

Do zakodowania opisu procesu biznesowego język BPEL wykorzystuję język znaczników XML.

8.4.1. Orkiestraция i choreografia usług

Wcześniej spotkaliśmy się z zagadnieniem choreografii usług. Polega ona na tym, że jedne usługi Web Services są klientami innych. Metoda ta wymaga od usług wiedzy jaką usługę wykonają w następnej kolejności.

Metoda orkiestracji usług jest uważana za bardziej elastyczną, ponieważ usługi składowe mogą uczestniczyć w wielu różnych złożonych procesach biznesowych bez konieczności modyfikacji ich kodu źródłowego.

Język BPEL umożliwia kompozycję złożonych procesów biznesowych z elementów prostych w oparciu o aranżację usług (orkiestracja). Zarządza on kolejnością wykonywania tych usług, przepływem informacji i reakcją na sytuacje awaryjne.
8.4.2. Proces biznesowy

Proces biznesowy w języku BPEL składa się z kroków zwanych akcjami. Akcje dzielą się na proste i złożone. Akcjami prostymi są:

- `invoke` – wywołanie usługi Web Service,
- `receive` – oczekiwanie na wywołanie procesu biznesowego przez klienta,
- `reply` – wygenerowanie odpowiedzi na wywołanie synchroniczne,
- `throw` – zasygnalizowanie awarii (wyjątek),
- `wait` – wstrzymanie wykonania na określony czas,
- `terminate` – zakończenie procesu biznesowego.

Akcje te mogą być składane w złożone algorytmy. Do sterowania przebiegiem wykonania algorytmu służą akcje:

- `sequence` – wykonanie akcji jedna po drugiej,
- `flow` – współbieżne wykonanie akcji,
- `switch` – rozgałęzienie sterowania (instrukcja warunkowa),
- `while` – pętla,
- `pick` – wybór ścieżki alternatywnej.

Ponadto język BPEL umożliwia definiowane zmiennych oraz powiązań z usługami. Skrypt BPEL komunikuje się ze światem zewnętrznym za pomocą `Partner Links` – odwołań do operacji zdefiniowanych w plikach WSDL.

8.4.3. NetBeans i graficzne edytory

Darmowe środowisko NetBeans po zainstalowaniu odpowiednich wtyczek umożliwia graficzną edycję plików BPEL, WSDL i XSD. Najlepiej w tym celu ściągnąć gotową instalację GlassFish ESB, która zawiera środowisko NetBeans z odpowiednimi wtyczkami oraz serwer aplikacji GlassFish z silnikiem BPEL. O tym skąd ściągnąć ten pakiet i jak go zainstalować jest napisane w instrukcji do poprzedniego laboratorium. Na poprzednim laboratorium było też omówione tworzenie projektu `BPEL Module` i `Composite Application` i nie będą one tutaj już omawiane.

Graficzny edytor plików BPEL bardzo ułatwia tworzenie procesów biznesowych. Edytor graficzny składa się z czterech widoków:

- `Source` – kod źródłowy pliku BPEL,
- `Design` – graficzna edycja procesu biznesowego,
- `Mapper` – graficzna edycja operacji na zmiennych (aby go waktywnić należy dwukrotnie kliknąć w jakąś akcję typu `Assign`),
- `Logging` – wypisanie wartości zmiennych do pliku logów serwera aplikacji, można się do niej przełączyć tylko z widoku `Mapper`.

Widok `Design` został przedstawiony na Rys. 10. Składa się on z trzech części. W lewej części znajdują się `Partner Links` wejściowe, pod które podpięty jest proces biznesowy. Natomiast z prawej strony znajdują się `Partner Links` operacji, których ten
proces biznesowy jest klientem. W środku natomiast znajdują się akcje przedstawione jako schemat blokowy.

Rys. 10. Graficzny edytor BPEL

W tym widoku dostępna jest paleta akcji. Aby dodać akcję do procesu biznesowego należy na palecie kliknąć odpowiednią akcję i przeciągnąć ją na schemat blokowy na któreś z kółeczek. Kółeczka te pojawiają się między umieszczonymi już akcjami w chwili przeciągania z palety nowej akcji.

Akcja Assign odpowiada za operacje na zmiennych. Główne zastosowanie to odpowiednie przekierowanie wejścia skryptu BPEL do wywołania innej usługi, a następnie przekierowania wyjścia tej usługi do wyjścia skryptu.

Silnik sun-bpel-engine umożliwia selektywne wybieranie pozycji ze złożonych struktur danych oraz proste operacje na liczbach i ciągach znaków.

Pozwala w prosty sposób edytować zawartości plików WSDL 2.0. Nie będziemy go wykorzystywać w naszych ćwiczeniach, ponieważ wszystko czego potrzebujemy ustawimy w kreatorze tworzenia nowego pliku WSDL.

Graficzny edytor XSD pozwala w prosty sposób zdefiniować struktury danych wykorzystywane jako parametry i wartości zwracane operacji zdefiniowanych w plikach WSDL. Jego działanie zostanie omówione w instrukcji tworzenia projektu Advanced Hello World.

8.4.4. Advanced Hello World w BPEL

Na poprzednich laboratoriach przedstawiony został opis tworzenia programu Hello World w BPEL. Nie poruszał on jednak tworzenia pliku XSD Schema, który tym razem zostanie omówiony.

Tworzymy nowy projekt z szablonu BPEL Module i nadajemy mu nazwę AdvBPEL.
Ponieważ dostęp do aplikacji BPEL odbywa się za pomocą interfejsu Web Services musimy stworzyć plik WSDL, który będzie zawierał interfejs aplikacji oraz plik XSD, który opisze używane przez nas struktury danych.

W laboratorium 7 tworzyliśmy prosty plik WSDL, którego zadaniem było wykorzystanie różnych komponentów BC. Tutaj stworzymy plik WSDL wykorzystujący bardziej zaawansowane struktury danych zdefiniowane w pliku XSD. Laboratorium 6 dotyczyło tworzenia Web Services, jednak pliki WSDL i XSD były tam generowane automatycznie na podstawie struktur danych i operacji udostępnianych przez aplikację. Tutaj mamy do czynienia z sytuacją odwrotną. Najpierw musimy określić struktury danych i operacje.

Aby otworzyć nowy plik XSD należy w środowisku NetBeans kliknąć prawym przyciskiem myszy na projekt AdvBPEL i z menu kontekstowego wybrać: New ➔ Other... ➔ XML ➔ XML Schema.

Nadajemy plikowi nazwę advBPEL (rozszerzenie .xsd zostanie dodane automatycznie). Po utworzeniu plik zostanie automatycznie otwarty do edycji. Edytor udostępnia trzy widoki:

- source,
- schema,
- design.

W tym ćwiczeniu będziemy pracować na widoku „schema”.

Przechodzimy następnie do utworzenia struktury złożonej, która zostanie użyta zarówno jako wejście jak i wyjście operacji. Strukturę złożoną tworzy się poprzez kliknięcie prawym przyciskiem na Complex Types i wybranie z menu kontekstowego Add Complex Type. Nowemu typowi złożonemu nadajemy nazwę „helloWorldType” i klikamy OK.

Rys. 12. Nowy element
Teraz należy do typu złożonego dodać co najmniej jeden element typu prostego lub innego wcześniej zdefiniowanego typ złożonego. W tym ćwiczeniu ograniczymy się do dodania jednego elementu typu prostego typu string. Aby tego dokonać należy kliknąć najpierw na Complex Types, następnie w drugiej kolumnie kliknąć na helloWorldType, a następnie w trzeciej prawym przyciskiem na sequence i z menu kontekstowego wybrać Add → Element. Następnie w oknie dialogowym należy zaznaczyć Use Existing Type, a z drzewa poniżej wybrać Build-in Types → string. Elementowi nadajemy nazwę paramA i klikamy OK.

Rys. 13. Graficzny edytor XSD

Na koniec musimy dodać element globalny do pliku XML Schema. W tym celu należy wrócić do kolumny pierwszej i kliknąć prawym przyciskiem myszy na Elements oraz z menu kontekstowego wybrać Add → Element. Następnie w oknie dialogowym wybrać Use Existing Type oraz w drzewie poniżej: Complex Types → helloWorldType. Nowemu elementowi nadajemy nazwę „typeA”.

Tworzymy nowy plik WSDL o nazwie advBPEL.wsdl. W polu WSDL Type wybieramy Concrete WSDL Document. Następnie pole Binding ustawiamy na SOAP, a pole Type na Document Literal i klikamy next. W zakładce, która się nam pojawia dokonujemy ustawień pierwszej operacji. Nazwę portu i operacji pozostawiamy bez zmian (nie będziemy dodawać nowych portów ani operacji potem). W polu Input w kolumnie Message Part Name wpisujemy „inputType”, a w kolumnie Element or Type klikamy na przycisk „…” i wybieramy: By File → AdvBPEL → src/advBPEL.xsd → Elements → typeA. Podobnie postępujemy z polem Output z tą różnicą, że w Message Part Name wpisujemy resultType. Następnie klikamy na next i potem na finish.

W końcu przechodzimy do pliku advBPEL.bpel który został utworzony wraz z projektem. Po otwarciu pliku upewniamy się, że w oknie edytora został włączony widok Design. Następnie metodą przyciagnij i upuść przeciągamy plik advBPEL.wsdl na kropkę znajdującą się na lewym marginesie widoku Design.

Z palety przeciągamy do głównego okna edytora BPEL trzy aktywności ustawiając je kolejno po sobie: Receive, Assign i Reply oraz podpinamy plik advBPEL.wsdl jako Partner Link. Proces umieszczania powyższych elementów jest identyczny jak ten z laboratorium 7.

W Assign łączymy paramA z AdvBPELOperationIn do paramA z AdvBPELOperationOut.
Zapisujemy projekt i klikamy na Build by sprawdzić czy się poprawnie skompiluje.

Na koniec tworzymy Composite Application w podobny sposób jak na poprzednich laboratoriach i dołączamy do niego projekt AdvBPEL. Następnie dla sprawdzenia działania tworzymy Test Case.

8.4.5. Integracja z usługami Web Services

W tym przykładzie nauczymy się jak zintegrować moduł BPEL z aplikacją udostępniającą swój interfejs w postaci interfejsu Web Services.

Wśród dołączonych do laboratorium aplikacji znajduje się aplikacja Hotel. Rezerwacja miejsc odbywa się za pomocą operacji booking, która na wejściu przyjmuje m.in. datę rozpoczęcia rezerwacji i datę końca rezerwacji. Załóżmy teraz, że klient życzy sobie by datę końca rezerwacji zamienić na liczbę dni. Mogli byśmy zmodyfikować sam program Hotel, ale kod został napisany przez niedoświadczonego studenta zatrudnionego na praktykę, który teraz wyjechał na wymianę studencką Erasmus, a kod aplikacji jest tak niedbale napisany, że za nic nie możemy się w nim połapać.

Z pomocą przychodzi język BPEL, który posłuży nam za tłumacza interfejsu wejścia.

Niestety operacje na danych w języku BPEL ograniczają się tylko do prostych porównań, prostych operacji arytmetycznych na liczbach i sklejania ciągów znaków. Nie mam możliwości wykonania skomplikowanych operacji na danych dlatego potrzebujemy napisać drugą aplikację Web Services, która udostępnia tylko jedną operację:

```
Date addDays(Date date, int days)
```

Dlaczego po prostu nie napiszemy nowej aplikacji Web Services, która będzie klientem aplikacji Hotel? Ponieważ chcemy by aplikacja DateCalculator była uniwersalna i niezależna od innych aplikacji i by można ją było bez modyfikacji jej kodu źródłowego użyć potem do innych celów.

Na tym etapie ćwiczeń zakładamy, że student ma w środowisku NetBeans otwarty projekt Hotel oraz napisał już aplikację DateCalculator z operacją addDays. Następnie stworzył nowy moduł BPEL o nazwie HotelBPEL, a w nim odpowiedni plik hotelBPEL.xsd – który definiuje nową strukturę danych wejścia i hotelBPEL.wsdl – który umożliwia dostęp do nowej aplikacji.

Plik hotelBPEL.xsd ma zawierać dwa elementy typu złożonego:

- Element hotel typu hotelType zawierający pojedyncze pola:
  - place typu string,
  - persons typu int,
  - start_date typu date,
  - days typu int.
- Element `hotelParagon` typu `hotelParagonType` zawierający pojedyncze pola:
  - `id` typu `int`,
  - `cost` typu `int`.

Plik `hotelBPEL.wsdl` ma zawierać *partner link* przyjmujący na wejściu element `hotel`, a na wyjściu zwracający element `hotelParagon`.

Powiązanie aplikacji z interfejsem Web Services z modułem BPEL polega na zainstalowaniu pliku WSDL aplikacji i stworzeniu odpowiedniego wrappera definującego `Partner Link` dla aplikacji.

NetBeans posiada kreator tworzenia takiego wrappera, który włącza się podczas próby użycia oryginalnego pliku WSDL aplikacji jako `Partner Link`. Aby go wywołać należy otworzyć plik `hotelBPEL.bpel`. Następnie w drzewie projektów rozwinąć `DateCalculator → Web Services`. Następnie kliknąć i przeciągnąć `DateCalculatorWS` na kropkę znajdującą się w środku prawej kolumny edytora pliku BPEL.

W oknie dialogowym nadajemy nazwę nowemu `Partner Link` "plDateCalculator". Resztę pozostawiamy bez zmian. Analogiczną operację wykonujemy dla aplikacji `Hotel`.

Do wykonania operacji wywołania służy akcja `Invoke`, która jest bardzo podobna do akcji `Receive` i `Reply`. Różni się tylko tym, że jest wiązana zarówno ze zmienną przechowującą jej wejście jak i ze zmienną przechowującą jej wyjście.

### 8.4.6. Zaawansowane konstrukcje BPEL


![Panel procesu biznesowego](image)

Rys. 14. Panel procesu biznesowego

Po zaznaczeniu całego procesu biznesowego, pojawiają się nad nim ikony umożliwiające zdefiniowanie dodatkowych właściwości procesu biznesowego. Jednym z nich jest operacja `Add Variable`...
Rys. 15. Definicja nowej zmiennej typu boolean o nazwie ifCond

Rys. 16. Akcja If w edytorze graficznym

Akcja If rozgałęzia przebieg wykonywania procesu biznesowego. Akcja If ma możliwość implementacji trzech typów gałęzi:

- gałąź główna (jeden raz),
- gałąź Else (jeden raz lub wcale),
- gałąź ElseIf (dowolna ilość).

Gałąź główna jest obowiązkowa i zawiera akcje, które zostaną wykonane jeżeli warunek jest spełniony. Gałąź ElseIf definiuje inny warunek, który gdy jest spełniony to wykonuje się ta gałąź ElseIf. W przeciwnym wypadku silnik BPEL
przechodzi do kolejnej gałęzi ElseIf lub gałęzi Else. Gałąź Else wykonuje się, gdy warunek główny ani warunek żadnej z gałęzi ElseIf nie został spełniony.

O ile akcja If rozdziela proces biznesowy na różne gałęzie, o tyle akcja While pozwala na powrót i powtórne wykonanie całego fragmentu procesu dowolną ilość razy. Przed wykonaniem pętli sprawdzany jest warunek. Jeżeli ma wartość true to pętla jest wykonywana, w przeciwnym wypadku następuje skok do końca akcji While. Po wykonaniu pętli, następuje skok na początek akcji While, gdzie znów sprawdzany jest warunek.

W odróżnieniu od pętli While pętla RepeatUntil wykonuje się dopóki warunek nie jest spełniony, czyli ma wartość false. Ponadto w pętli RepeatUntil warunek sprawdzany jest po wykonaniu ciała pętli. Gwarantuje to, że pętla wykonana będzie co najmniej raz. Z kolei pętla ForEach wykonuje się zadaną ilość razy.

Rys. 17. Akcja While w edytorze graficznym

Rys. 18. Akcja Flow w edytorze graficznym

Akcja Flow rozdziela wykonanie procesu biznesowego na wiele gałęzi. Każda z gałęzi może wykonywać się współbieżnie. Akcja ta kończy się, gdy wszystkie

![Diagram](image)

Rys. 19. Akcja Pick w edytorze graficznym

Akcja Pick przydaje się, gdy chcemy np. zaimplementować obsługę *TimeOut* podczas oczekiwania na wiadomość asynchronicznego wywołania jakiejs usługi.


Jak wiadomo, wykonanie usługi nie koniecznie musi zakończyć się powodzeniem. Usługi zgłaszają błąd wykonania przez rzucając wyjątek (*Throw*). Wyjątek jest to obiekt odpowiedniej klasy zawierający informacje o błędzie. Sam typ obiektu jest już sporą informacją o typie błędu (np. *FileNotFoundException*). Dodatkowo zawartość obiektu pozwala poznać więcej szczegółów.

BPEL umożliwia obsługę wyjątków za pomocą konstrukcji *Catch* i *Catch All*. Konstrukcje *Catch* dodaje się za pomocą panelu procesu biznesowego przedstawionego na Rys. 14.

Konstrukcja *Catch* łapie kolejny wyjątek – odpowiedni *FaultMessage* zdefiniowany w pliku WSDL. Obiekt *FaultMessage* możemy przechwyścić i umieścić w zmiennej powiązanej z konstrukcją *Catch*, co umożliwia odczytanie jego zawartości.
Konstrukcja Catch All łapie wszystkie wyjątki. Ponieważ złapany wyjątek jest nieznanego typu, nie jest możliwe odczytanie jego zawartości.

Konstrukcje Catch nie dotyczą tylko całych procesów biznesowych. Mogą łapać jedynie fragment procesu biznesowego. Taki fragment wydziela się z procesu biznesowego za pomocą konstrukcji Scope, która jest jakby wydzielonym podprocesem biznesowym.

8.5. Przebieg laboratorium

Druga część instrukcji zawiera zadania do praktycznej realizacji, które demonstrują zastosowanie technik z omawianego zagadnienia. Podczas realizacji zadań należy zapoznać się z kodem źródłowym BPEL. Należy zaobserwować, jakie zmiany wprowadza w nim każda operacja w graficznym edytorze.

8.5.1. Przygotowanie laboratorium

Programy opisane w zadaniach zostaną napisane w języku Java w środowisku NetBeans. Przed przystąpieniem do rozwijywania zadań należy upewnić się, że w środowisku został zainstalowany Java Developer Kit, NetBeans z odpowiednimi wtyczkami oraz serwer aplikacji Glassfish ESB.

Do wykonania zadań opisanych w tym laboratorium potrzebne są aplikacje Hotel, Airport i Bank znajdujące się w dołączonych materiałach.
8.5.2. Zadanie 1. Prowizja

W dołączonych materiałach do laboratorium znajduje się aplikacja Hotel. Rezerwacja miejsc odbywa się za pomocą operacji booking, która na wejściu przyjmuje m.in. liczbę rezerwowanych miejsc, a na wyjściu cenę. Teraz potrzebujemy napisać proces biznesowy dla pośrednika, który do tej ceny będzie naliczał prowizję. Prowizja ma wynosić 5% za rezerwację mniej niż 3 miejsce, 3% za rezerwację mniej niż 10 miejsc i 1% za rezerwację pozostałych ilości. Wejście ma zawierać te same pola, co wejście operacji booking aplikacji Hotel. Wyjście natomiast ma zawierać te same pola, co wyjście operacji booking aplikacji Hotel oraz dodatkowe wyjście „provision” zawierające wysokość prowizji.

Zadanie ma celu przećwiczyć złożone typy danych definiowane w pliku XSD, akcję If oraz operacje arytmetyczne na liczbach w języku BPEL.

8.5.3. Zadanie 2. Prototyp biura podróży

Napisać proces biznesowy, który na wejściu przyjmie dane wycieczki. Zarezerwuje wycieczkę w Hotel, kupi odpowiednie bilety w Airport i obciąży konto w Bank. Należy skorzystać z aplikacji z materiałów do laboratorium.

Zadanie ma na celu przećwiczyć sekwencje równoległe w języku BPEL oraz pracę z bardziej rozbudowanymi procesami biznesowymi.