Biuletyn WAT
Vol. LXVII, Nr 3, 2018

Transformacja diagramu aktywności UML w przepływ integracyjny BPEL

TOMASZ GÓRSKI, GRZEGORZ ZIEMSKI

Akademia Marynarki Wojennej, Instytut Uzbrojenia Okrętowego i Informatyki,
ul. Śmidowicza 69, 81-127 Gdynia
t.gorski@amw.gdynia.pl, ziemski@gmail.com

Streszczenie. Rosnące zainteresowanie firm integracją oraz interoperacyjnością systemów informatycznych spowodowało wzrost znaczenia architektury usługowej (ang. *Service-Oriented Architecture*), która zapewnia narzędzia umożliwiające integrację aplikacji korporacyjnych (ang. *Enterprise Application Integration*). W tym sensie magistrala usług (ang. *Enterprise Service Bus*) zapewnia techniczne możliwości komunikacji między systemami informatycznymi. Kluczowym elementem w tej komunikacji są przepływy integracyjne.
Cel: Celem artykułu jest przedstawienie nowej transformacji Integration2BPEL, która automatyzuje konstrukcję wykonywalnego przepływu integracyjnego wyrażonego w języku Web Services Business Process Execution Language (BPEL) na podstawie modelu tego przepływu przedstawionego na diagramie aktywności języka Unified Modeling Language (UML).
Metoda: Autorzy proponują transformację typu model-to-code generującą przepływ integracyjny wyrażony w BPEL, który może być uruchamiany w dowolnym silniku procesów BPEL. Przepływ integracyjny modelowany jest za pomocą diagramu aktywności języka UML z użyciem stereotypów z profilu „UML Profile for Integration Flows" w środowisku IBM Rational Software Architect (RSA). Przy zastosowaniu transformacji Integration2BPEL generowany jest kompletny, wykonywalny przepływ integracyjny złożony z wielu mechanizmów mediacyjnych. Wygenerowany przepływ integracyjny uruchamiany był na magistrali usług OpenESB.
Wyniki: Możliwość generacji kompletnego przepływu integracyjnego w BPEL, który bez żadnych uzupełnień może być uruchamiany na magistrali usług. Zautomatyzowana została faza implementacji przepływu integracyjnego. Każdy z przepływów integracyjnych implementowany jest według takich samych zasad. Ponadto, unika się dzięki temu błędów popełnianych przez projektantów i programistów.
Wnioski: Wytwarzanie oprogramowania sterowane modelami (ang. *Model-Driven Development*) jest podejściem, które może prowadzić do automatyzacji fazy projektowania i programowania. Uzyskuje się wprowadzenie jednolitego mechanizmu konstrukcji przepływu integracyjnego.

Słowa kluczowe: Web Services Business Process Execution Language (BPEL), Enterprise Service Bus (ESB), Unified Modelling Language (UML), Diagram aktywności UML, Model-Driven Development (MDD), transformacje
DOI: 10.5604/01.3001.0012.6587

1. Wprowadzenie

Głównym celem interoperacyjności jest zapewnienie odpowiednich mechanizmów dla integracji różnych systemów informatycznych. Architektura usługowa (ang. *Service-Oriented Architecture – SOA*) dostarcza mechanizmów do integracji aplikacji w przedsiębiorstwach (ang. *Enterprise Application Integration*). Wiodące technologie do realizacji SOA wykorzystują standardy WS-*, takie jak SOAP, WSDL i BPEL. Magistrala usług (ang. *Enterprise Service Bus*) pojawiła się w celu umożliwienia integracji i współdziałania różnych aplikacji, usług lub innych zasobów w organizacjach [1]. W tym sensie magistrala usług odgrywa kluczową rolę jako technologia leżąca u podstaw tworzenia zintegrowanych środowisk biznesowych, w których firmy mogą połączyć wszystkie swoje aplikacje. W rzeczywistości ESB dostarcza warstwę mediacyjną dla dostępu i przetwarzania informacji do kilku protokołów, umożliwiającą komunikację między różnymi aplikacjami zintegrowanymi na ESB za pomocą adapterów.

Duże znaczenie tej technologii spowodowało, że zostały opracowane komercyjne magistrale usług, na przykład IBM Integration Bus [2], TIBCO Business Works [3] i Oracle Service Bus [4]. Ponadto powstały implementacje magistrali usług typu Open Source takie jak JBoss Fuse [5], Mule ESB [6], Petals ESB [7], WSO2 Enterprise Service Bus [8] i OpenESB [9]. Wiele firm i projektów badawczych wykorzystuje magistrale usług typu Open Source [10]. Ponadto, dostawcy magistral usług typu Open Source poczynili znaczne postępy w stosunku do komercyjnych dostawców ESB i teraz stanowią realną alternatywę dla coraz większej liczby przedsiębiorstw [11]. Dodatkowo, zastosowanie magistrali typu Open Source w rozwiązaniach integracyjnych może prowadzić do oszczędności w postaci redukcji kosztów zakupu platformy wykonawczej ESB oraz kosztów utrzymania [12]. Ponadto, ważną zaletą magistral usług typu Open Source jest to, że integrują systemy informatyczne za pomocą otwartych protokołów i standardów. Coraz szersze zastosowanie oprogramowanie typu Open Source znajduje także w administracji publicznej [13]. W artykule wykorzystano magistralę usług Open ESB. Jednym z takich otwartych standardów stosowanych w integracji systemów informatycznych jest język BPEL. Język ten jest, opartą na standardach, technologią orkiestracji usług, która dostarcza gramatyki opartej na języku XML opisującej logikę sterowania wymaganą do orkiestracji usług sieciowych uczestniczących w przepływie integracyjnym. Przepływy integracyjne określone w BPEL są przenośne i mogą być wykonywane w dowolnym silniku procesu zgodnym z BPEL.

Przedmiotem artykułu jest opracowana transformacja Integration2BPEL automatyzacji budowy przepływów integracyjnych z wykorzystaniem modeli UML (ang. *Unified Modeling Language*) [14, 15]. Utworzone oprogramowanie umożliwia transformację typu model-to-code modelu przepływu integracyjnego wyrażonego na diagramie przepływu integracyjnego UML w wykonywalny przepływ integracyjny wyrażony w języku BPEL (ang. *Business Process Execution Language for Web Services*) [16]. Diagram przepływu integracyjnego UML stanowi rozszerzenie diagramu aktywności języka UML z zastosowanymi stereotypami z profilu UML Profile for Integration Flows [17]. Przepływy integracyjne są jednym z wielu elementów opisu architektonicznego platformy integracyjnej. Model widoków architektonicznych dostosowany do potrzeb modelowania platform integracyjnych oraz zestaw konstrukcji modelowych, dzięki któremu można przedstawić pełną architekturę platformy integracyjnej, przedstawiono w literaturze [18].

Artykuł został ułożony w przedstawiony dalej sposób. W sekcji 2 omawiane jest zagadnienie przepływów integracyjnych i wzorców mediacyjnych. W sekcji 3 opisano elementy języka BPEL, które powstają w wyniku działania transformacji Integration2BPEL. Sekcja 4 stanowi przegląd prac powiązanych tematycznie z integracją systemów informatycznych i zastosowaniem inżynierii sterowanej modelami. W sekcji 5 zaprezentowano przykład, przy którym została użyta transformacja Integration2BPEL. W sekcji 6 przedstawiono część funkcjonalną — zasadę działania transformacji Integration2BPEL z uwzględnieniem mapowania między elementami UML i BPEL. Opisane zostały przykładowe fragmenty kodu UML oraz fragmenty kodu BPEL, który powstaje w wyniku transformacji. Sekcja 7 stanowi część techniczną — wprowadzenie w implementację transformacji Integration2BPEL. Sekcja 8 to uzasadnienie wyboru technologii do utworzenia transformacji. Podsumowanie i kierunki dalszych prac przedstawione zostały w sekcji 9.

2. Przepływy integracyjne i wzorce mediacyjne

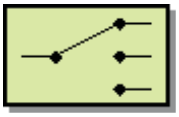
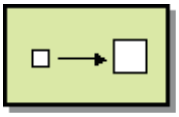
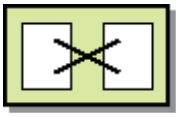
Przepływy integracyjne stosowane są do przesyłania danych między różnymi systemami informatycznymi. W celu zaspokojenia rosnącego zapotrzebowania na aktualne dane wiele komunikatów jest wymienianych między systemami. Efektywność tych przepływów integracyjnych jest zatem istotna, aby poradzić sobie z dużym obciążeniem wymiany wiadomości oraz zmniejszyć opóźnienia w ich dostarczaniu. Technicznie odnosi się to do zdolności dwóch lub większej liczby systemów do wymiany informacji i możliwości użycia tych wymienionych informacji.

W przepływach integracyjnych kluczową rolę odgrywają wzorce mediacyjne. Są one wykorzystywane do odbioru komunikatu, następnie wykonania określonych działań i przekazania komunikatu do odbiorcy. Zarówno odbiór, jak i przekazywanie dalej komunikatów może odbywać się różnymi, zależnymi od wybranych

technologii, kanałami (np. FTP, HTTP, Message Queue). Wzorce mediacyjne to zbiór sprawdzonych i przetestowanych dobrych praktyk zastosowanych w trakcie integracji [19]. Każdy wzorzec określa podstawową czynność, jaką wykonuje w przepływie integracyjnym. Jednym z najprostszych wzorców mediacyjnych jest kanał wiadomości, którego rola ogranicza się do przesłania otrzymanego komunikatu od wywołującego do odbiorcy. W tabeli 1 przedstawiono wybrane wzorce mediacyjne wraz z opisem, które zostały wykorzystane w dalej zaprezentowanym przykładzie.

Tabela 1

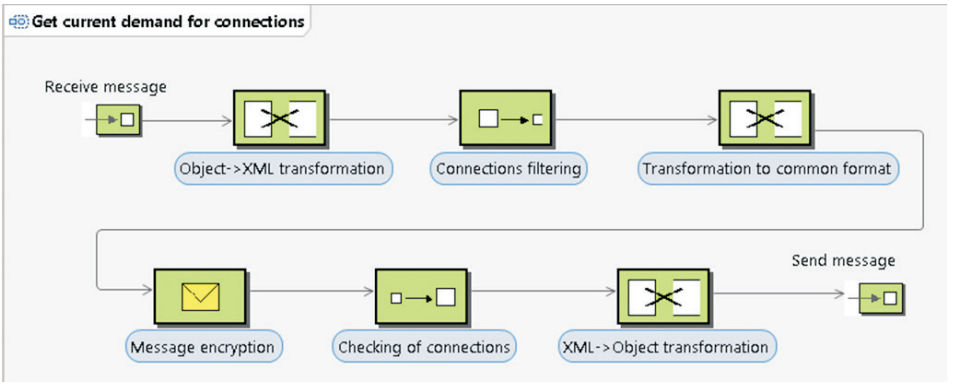
Wybrane wzorce mediacyjne

Wzorzec	Reprezentacja graficzna	Opis wzorca
Kanał wiadomości, ang. *Channel*		Określa, w jaki sposób dostawca komunikuje się z klientem. Jest to najprostszy wzorzec mediacyjny.
Publikuj subskrybuj, ang. *Publish subscribe*		Powiela otrzymany komunikat, tak aby każdy ze subskrybentów otrzymał jego kopię. Zapewnia, że komunikat zostanie otrzymany przez każdego z odbiorców tylko raz.
Trasowanie oparte o treść, ang. *Content based router*		Określa, do którego odbiorcy ma trafić komunikat w zależności od jego treści. Warunki trasowania są określone przez konfigurację przepływu.
Wzbogacanie treści, ang. *Content enricher*		Określa, w jaki sposób zostanie wzbogacona treść przetwarzanego komunikatu.
Translacja wiadomości, ang. *Message Translator*		Wzorzec konwertuje format wiadomości systemu źródłowego na format wiadomości systemu docelowego.

Po to, aby móc modelować przepływy integracyjne, zaproponowano diagram przepływów integracyjnych [17], który stanowi rozszerzenie diagramu aktywności języka UML. Diagram przepływów integracyjnych stosowany jest w widoku integrowanych usług w modelu widoków architektonicznych „1 + 5" [18]. Na diagramie tym modelowany jest kompletny przepływ integracyjny. Przykład diagramu przepływów integracyjnych z zastosowaniem stereotypów z profilu 'UML Profile for Integration Flows' zaproponowanego przez Górskiego [17] został przedstawiony

na rysunku 1. Artykuł przedstawia transformację Integration2BPEL generującą wykonywalny przepływ integracyjny w języku BPEL [16]. Diagram przepływów integracyjnych jest wystarczająco szczegółowy, aby móc wygenerować wykonywalny przepływ integracyjny. Jako platformę wykonawczą przyjęto magistralę usług typu Open Source OpenESB.

Transformacja Integration2BPEL jest wertykalna, co oznacza, że elementy źródłowy i docelowy są na różnych poziomach abstrakcji. Ponadto, transformacja ta jest egzogenna, ponieważ elementy źródłowy i docelowy wyrażone są w różnych notacjach: UML oraz BPEL. W obecnej postaci jest to także transformacja jednokierunkowa. Kompletna klasyfikacja transformacji została przedstawiona przez Mens [20].



Rys. 1. Przepływ integracyjny dla wywołania usługi Get current demand for connections

3. BPEL – Business Process Execution Language

BPEL [38] jest językiem przeznaczonym do definiowania wykonywalnych oraz abstrakcyjnych procesów biznesowych wykorzystującym usługi sieciowe. BPEL to język orkiestracji usług sieciowych, a nie choreografii. Podstawowa różnica polega na tym, że w choreografii usługi współpracują ze sobą bezpośrednio. W orkiestracji usługa komunikuje się wyłącznie z menedżerem orkiestracji (w tym przypadku: maszyna BPEL), który wie, gdzie i jak przekazać dalej komunikaty. Nie musi tym samym znać innych usług biorących udział w procesie. Procesy wykonywalne BPEL są interpretowane i wykonywane przez maszyny procesów biznesowych w środowisku zgodnym ze standardem BPEL. Standard rozwijany jest przez OASIS (ang. *Organization for the Advancement of Structured Information Standard*) [39]. Język oparty jest na XML (ang. *Extensible Markup Language*), BPEL opisuje zachodzące interakcje, steruje przepływem pracy i przekazywaniem danych. Można wyróżnić dwa typy procesów (przepływów integracyjnych): wykonywalny i abstrakcyjny.

Proces wykonywalny opisuje powiązane ze sobą działania, które prowadzą do osiągnięcia określonego efektu. Każde działanie opisane jest informacjami technicznymi o sposobie wykonania danego działania. Poprawnie zdefiniowany proces wykonywalny może być zinterpretowany przez silnik procesów i wykonany. Proces abstrakcyjny pełni rolę opisową, pozbawiony jest informacji technicznych, opisuje proces z punktu widzenia wymagań biznesu. W artykule koncentrujemy się na procesach wykonywalnych.

W tabeli 2 przedstawiono wybrane elementy języka BPEL.

Tabela 2

Wybrane elementy BPEL

Element	Opis elementu
PartnerLink	Definiuje usługę.
Invoke	Wywołuje metodę usługi sieciowej. W definicji elementu należy podać nazwę metody oraz nazwę PartnerLink dla usługi. Zostaje wysłana i odebrana wiadomość zdefiniowana w elemencie Variable.
Receive	Port wejściowy, nasłuchujący otrzymanie żądania.
Reply	Port wyjściowy, zwraca komunikat do elementu wywołującego (Partnera).
Variable	Zmienna przechowuje komunikat przesyłany lub odbierany z usługi.
Assign	Przypisywanie danych do zmiennych. Wykorzystując tę operację, można wykonać operacje z kategorii wzorców mediacyjnych – transformacja wiadomości.
Sequence	Zbiór operacji do sekwencyjnego wykonania.
If-else	Rozgałęzienie przebiegu przepływu. W zależności od spełnienia warunku przepływ integracyjny wybiera jedną ze ścieżek.

W języku BPEL można wyróżnić pięć głównych sekcji:

— przepływ wiadomości (ang. *message flow*) — wywoływanie operacji na usługach sieciowych, przesyłanie żądania i oczekiwanie na wynik,
— przepływ sterowania (ang. *control flow*) — sposób występowania po sobie działań w przepływie,
— przepływ danych (ang. *data flow*) — zmienne używane w procesie biznesowym,
— orkiestracja procesów (ang. *process orchestration*) — ustanawia zależności między usługami,
— obsługa błędów i wyjątków (ang. *fault and exception handling*) — opisuje, jak należy obsługiwać błędy, które mogą się pojawić w trakcie wykonywania procesu.

Na poziomie technicznym rola języka BPEL rozpoczyna się tam, gdzie kończy WSDL (ang. *Web Services Description Language*) [41], który opisuje interfejs pojedynczej usługi. BPEL opisuje, jak zbiór usług wystawianych przez różne aplikacje

może współpracować ze sobą do osiągnięcia konkretnego celu biznesowego. Każdy proces BPEL również jest usługą sieciową, dlatego także opisywany jest przez WSDL i może być używany jako część składowa innych procesów. Do konstruowania warunków w elementach sterujących oraz konstrukcjach przypisywania do zmiennych wykorzystywany jest XPath (ang. *XML Path Language*) [42]. Wykorzystując XPath, możliwe jest adresowanie do poszczególnych elementów w XML.

4. Prace powiązane tematycznie

Artykuł dotyczy dwóch istotnych zagadnień: architektury usługowej [21], w szczególności platform integracyjnych [22], oraz wytwarzania oprogramowania sterowanego modelami (ang. *Model-Driven Development – MDD*) [23]. Zastosowanie MDD w rozwiązaniach w architekturze usługowej jest obiektem aktualnych badań [24, 25, 26]. Jednym z wniosków przeglądu literatury jest stwierdzenie znaczącego procentu badań bez wsparcia narzędziowego [24]. W tym kontekście nasz artykuł wnosi kompletne rozwiązanie ze wsparciem narzędziowym IBM RSA oraz Open ESB i implementacją transformacji w języku Java. Ponadto, Ameller [24, s. 53] wskazuje, że większość publikowanych badań jest realizowana w narzędziach Eclipse (41,7%) oraz IBM Rational (20,8%). W przeglądzie tym [24, s. 53] podano także, że większość elementów źródłowych transformacji w badaniach stanowią modele UML (41,7%), a elementów docelowych pliki BPEL (30,6%). W świetle tych danych zaproponowana przez nas transformacja Integration2BPEL wpisuje się w nurt aktualnych badań.

Zagadnienie transformacji modeli wyrażonych w BPMN i BPEL jest aktualne w literaturze. W [28] przedstawiono zagadnienie transformacji dwukierunkowej między modelami BPMN i BPEL. Dostępne są także prace pokazujące transformacje modeli przekształcające modele BPMN w modele UML. Natomiast dotyczą one transformacji procesów biznesowych wyrażonych w BPMN w diagramy aktywności języka UML [29, 30]. W literaturze przedmiotu znajdują się także przykłady transformacji modeli UML w przepływy BPEL [31, 32]. Dotyczą one jednak modeli UML wyrażających procesy biznesowe, a nie przepływy integracyjne. Ponadto transformacja [31] nie może być samodzielnie uruchamiana, gdyż jest częścią składową transformacji „UML to SOA" [33]. Spotyka się także rozwiązania pokazujące translacje innych formatów zapisu modeli procesów biznesowych. Jednym z nich jest przekształcenie modeli GLIF w modele XPDL [34].

W artykule koncentrujemy się na transformacji diagramów aktywności UML w diagramy przepływów integracyjnych BPEL. Transformacja ta wpisuje się w model widoków architektonicznych „1 + 5" [18]. W ramach tego modelu przedstawiono wcześniej następujące transformacje [35]: widoku integrowanych procesów do widoku przypadków użycia (BPMN2UC), widoku przypadków użycia do widoku

logiki (UC2Logical), widoku przypadków użycia do widoku integrowanych usług (UC2IS), widoku przypadków użycia do widoku kontraktów (UC2Contracts). W obszarze widoku integrowanych usług przedstawiono także transformację [36] modelu przepływu integracyjnego wyrażonego w postaci diagramu aktywności UML w kod przepływu integracyjnego w języku Java w postaci kompletnej aplikacji klasy enterprise osadzanej na serwerze aplikacyjnym IBM Enterprise Service Bus. Transformacja ta dotyczyła pojedynczych mechanizmów mediacyjnych i pokazywała implementację przepływu w środowisku komercyjnej magistrali usług. Natomiast transformacja Integration2BPEL generuje kompletny przepływ integracyjny złożony z wielu mechanizmów mediacyjnych oraz generuje przepływ w języku BPEL, który jest stosowany w magistralach usług typu Open Source. W obu wymienionych wyżej transformacjach stosowano do modelowania środowiska IBM RSA. W obszarze modelowania procesów biznesowych, szczególnie w notacji BPMN, stosuje się także IBM WebSphere Business Modeler [37].

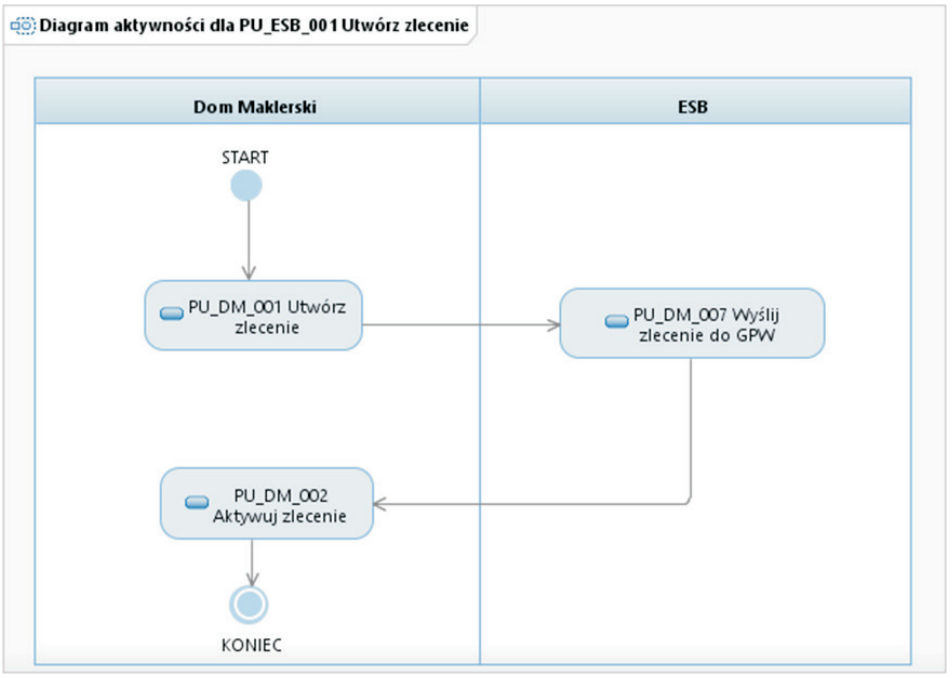
5. Przypadek użycia *Utwórz zlecenie*

Transformacja Integration2BPEL została zastosowana do przypadku użycia *Utwórz zlecenie* w ramach aplikacji dla Domu Maklerskiego. W ramach realizacji przypadku użycia *Utwórz zlecenie* współpracują ze sobą systemy Giełdy Papierów Wartościowych (GPW) i Domu Maklerskiego przy wykorzystaniu Platformy Integracyjnej. System informatyczny GPW umożliwia obrót giełdowy papierami wartościowymi. Transakcja kupna-sprzedaży w systemie informatycznym GPW zostaje zawarta, gdy na rynku obrotów pojawią się dwa przeciwstawne zlecenia kupna i sprzedaży dla tej samej spółki, gdzie kupujący zgodzi się zapłacić cenę zaproponowaną przez sprzedającego. Wymianę danych pomiędzy systemem GPW a systemem Domu Maklerskiego umożliwia Platforma Integracyjna. Przepływ zdarzeń przypadku użycia *Utwórz zlecenie* obejmuje: utworzenie zlecenia, wysłanie zlecenia do GPW oraz aktywowanie zlecenia. Przepływ ten przedstawiony został na diagramie aktywności języka UML (rysunek 2).

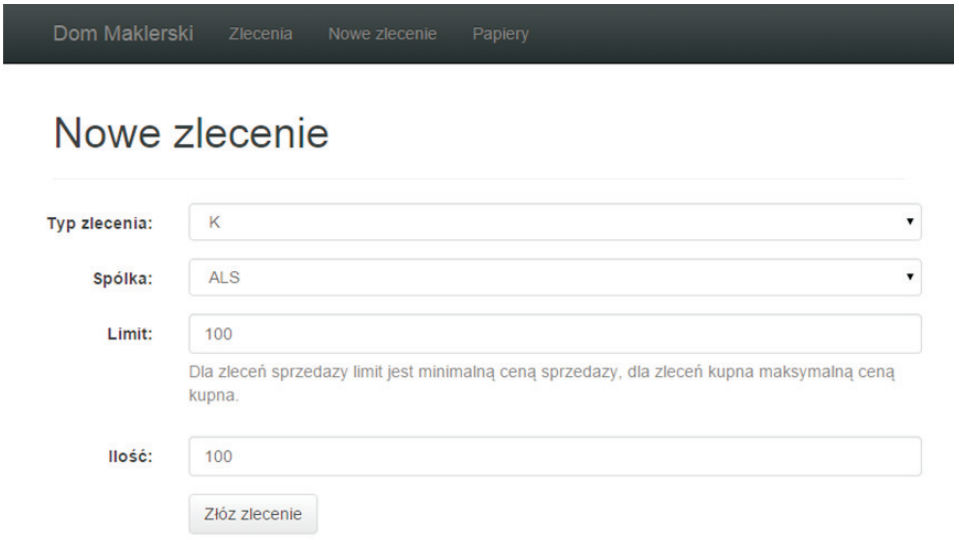
Pierwszą aktywność w przepływie stanowi PU_DM_001 *Utwórz zlecenie*. W ramach tej aktywności inwestor wysyła wypełniony formularz. Zlecenie zapisywane jest w systemie Domu Maklerskiego ze statusem „OCZEKUJĄCE".

Rysunek 3 przedstawia projekt formatki do wprowadzania nowego zlecenia.

Druga aktywność PU_DM_007 *Wyślij zlecenie do GPW* realizuje przepływ integracyjny pomiędzy Domem Maklerskim i GPW na Platformie Integracyjnej (ESB). Przepływ integracyjny *Wyślij zlecenie do GPW* został przedstawiony na diagramie aktywności (rysunek 4). Na podstawie tego typu diagramów UML tworzone są przepływy integracyjne w BPEL. Na diagramie tym, utworzonym w środowisku IBM Rational Software Architect, zastosowano stereotypy z profilu UML *Profile for Integration Flows*.



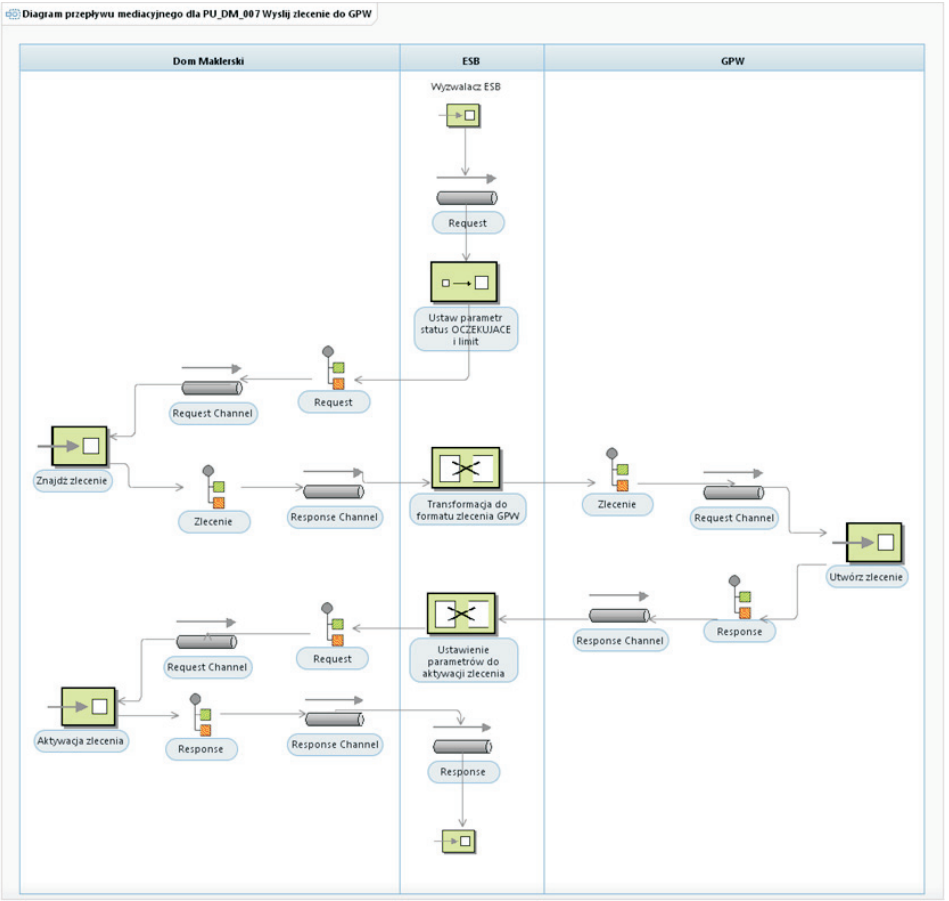
Rys. 2. Diagram aktywności dla PU_ESB_001 *Utwórz zlecenie*



Rys. 3. Projekt formatki dla PU_DM_001 *Utwórz zlecenie*

Aktywność PU_DM_007 *Wyślij zlecenie do GPW* składa się z następujących akcji:

- Przepływ uruchamia czasowy wyzwalacz na ESB.
- Ustaw parametr status OCZEKUJĄCE i limit — tworzony jest komunikat z parametrami wyszukiwania zleceń.
- Znajdź zlecenie — wyszukiwanie zlecenia w Domie Maklerskim spełniającym warunki wyszukiwania.
- Transformacja do formatu zlecenia GPW — transformacja komunikatu zlecenia z formatu Domu Maklerskiego do formatu wspieranego przez GPW.
- *Utwórz zlecenie* — zostaje utworzone identyczne biznesowo zlecenie w GPW.
- Ustawienie parametrów do aktywacji zlecenia — przygotowanie komunikatu aktywacji zlecenia w Domu Maklerskim.
- Aktywacja zlecenia — zlecenie w systemie Domu Maklerskiego zmienia status z OCZEKUJĄCE na AKTYWNE.



Rys. 4. Diagram przepływu integracyjnego *Wyślij zlecenie do GPW*

Trzecia aktywność PU_DM_002 *Aktywuj zlecenie* zmienia rejestr aktywów inwestora oraz stan jego środków pieniężnych. Zlecenie otrzymuje status AKTYWNE.

Rysunek 5 przedstawia projekt formatki do przeglądania zleceń. Na tym rysunku przedstawiono jedno zlecenie, którego złożenie zostało potwierdzone przez GPW.

Rys. 5. Projekt formatki dla PU_DM_005 Przeglądaj zlecenia

Analiza efektywności transformacji Integration2BPEL

W celu oszacowania efektywności stosowania transformacji Integration2BPEL wykonano kompletny przepływ integracyjny w BPEL na dwa sposoby: ręcznie oraz z zastosowaniem transformacji. Niezależnie od wybranego sposobu uzyskania przepływu integracyjnego w BPEL należy wykonać następujące kroki:

- utworzenie diagramu klas,
- utworzenie diagramu aktywności modelu przepływu integracyjnego w UML,
- uzupełnienie transformacji komunikatów do modelu UML,
- utworzenie przepływu integracyjnego w BPEL: ręcznie albo za pomocą transformacji.

W dwóch pierwszych krokach powstaje diagram klas i aktywności w UML. W kroku trzecim programista BPEL/OpenESB przygotowuje BPEL albo uruchamia transformację. Aby BPEL był w pełni wykonywalny, należy wcześniej w UML uzupełnić kody transformacji komunikatów. Trzeba to zrobić w modelu przepływów integracyjnych w UML, opisując transformację w XML. W OpenESB istnieje do tego specjalnie przygotowane narzędzie graficzne. Przy zastosowaniu transformacji zaobserwowano ok. pięciokrotne skrócenie czasu utworzenia przepływu integracyjnego w BPEL z diagramu aktywności języka UML.

6. Działanie transformacji Integration2BPEL

a. Przygotowanie modelu źródłowego

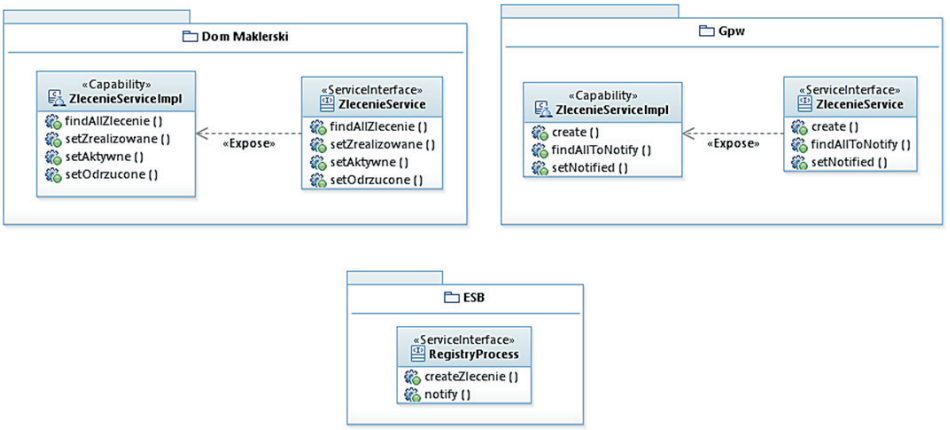
Dla transformacji Integration2BPEL modelem źródłowym jest model UML rozszerzony o profil „UML Profile for Integration Flows". Przygotowany model źródłowy UML musi posiadać zdefiniowany widok Integrowanych Usług, a w nim dwa typy diagramów:

— diagramy klas, na których zostały opisane interfejsy używanych usług sieciowych,
— diagramy aktywności, reprezentujące przepływy integracyjne.

Przykładowy diagram klas z pokazanymi interfejsami używanych usług sieciowych przedstawiono na rysunku 6. Do opisu użyte zostały stereotypy z SoaML [43], które w kontekście tego diagramu oznaczają:

— ServiceInterface — interfejs do komunikacji, definicja usługi internetowej,
— Capability — obiekt implementujący funkcjonalność,
— Expose — relacja wskazująca, które obiekty implementują funkcjonalności dla wybranej usługi sieciowej.

Przykład diagramu przepływu integracyjnego utworzonego w środowisku IBM Rational Software Architect z wykorzystaniem stereotypów z profilu „UML Profile for Integration Flows" przedstawiono na diagramie aktywności UML (rysunek 4). Diagram ten opisuje przypadek PU_DM_007 *Wyślij zlecenie do GPW*.



Rys. 6. Diagram klas z określonymi interfejsami usług sieciowych

b. Zainicjowanie procesu BPEL

Wykonywanie BPEL można rozpocząć przez wywołanie metody usługi sieciowej zgodnej ze standardem WS-BPEL. Metoda ta może przyjmować argumenty, które są danymi wejściowymi dla przepływu, jednocześnie może zwrócić dane wyjściowe. Istotne jest, aby dla BPEL-a został opisany interfejs w WSDL (Listing 1) wraz z omawianą metodą umożliwiającą uruchamianie procesu.

Listing 1. Interfejs procesu BPEL zapisany w WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="realizujZlecenia"
 targetNamespace="http://j2ee.netbeans.org/wsdl/gpw-openesb-bpel/src/realizujZlecenia"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://j2ee.netbeans.org/wsdl/gpw-openesb-bpel/src/realizujZlecenia"
 xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype">
 <types/>
 <message name="realizujZleceniaOperationRequest">
    <part name="part1" type="xsd:string"/>
 </message>
 <message name="realizujZleceniaOperationResponse">
    <part name="part1" type="xsd:string"/>
 </message>
 <portType name="realizujZleceniaPortType">
    <operation name="realizujZleceniaOperation">
       <input name="input1" message="tns:realizujZleceniaOperationRequest"/>
       <output name="output1" message="tns:realizujZleceniaOperationResponse"/>
    </operation>
 </portType>
 <plnk:partnerLinkType name="realizujZlecenia">
   <plnk:role name="realizujZleceniaPortTypeRole" portType="tns:realizujZleceniaPortType"/>
 </plnk:partnerLinkType>
</definitions>
```

W modelu UML elementem rozpoczynającym przepływ musi być InitialNode ze stereotypem Endpoint. Przykładowe fragmenty kodu UML (Listing 2) zostały opisane dla „Wyzwalacz ESB" (rysunek 4). InitialNode trzeba połączyć relacją typu Usage z metodą w interfejsie, która będzie rozpoczynać przepływ.

Listing 2. Fragmenty kodu UML elementu rozpoczynającego przepływ mediacyjny „Wyzwalacz ESB”

```
<!-- Element UML rozpoczynający przypływ -->
<node xmi:type=”uml:InitialNode”
    xmi:id=”_kUIeZ6DtEeSJP70eRk6eCg”
    name=”Wyzwalacz ESB”
    clientDependency=”_kUN9XKDtEeSJP70eRk6eCg”
    outgoing=”_kUIelaDtEeSJP70eRk6eCg”/>
...
<!-- Metoda w usłudze sieciowej rozpoczynająca przypływ -->
<ownedOperation xmi:id=”_kUN9WaDtEeSJP70eRk6eCg” name=”createZlecenie”/>
...
<!-- Relacja pomiędzy Endpoint a metodą w usłudze sieciowej -->
<packagedElement xmi:type=”uml:Usage”
            xmi:id=”_kUN9XKDtEeSJP70eRk6eCg”
            name=””
            supplier=”_kUN9WaDtEeSJP70eRk6eCg”
            client=”_kUIeZ6DtEeSJP70eRk6eCg”/>
...
<!-- Stereotyp Endpoint dodany do InitialNode -->
<EIP:Endpoint xmi:id=”_kUN9cqDtEeSJP70eRk6eCg”
        base_ActivityNode=”_kUIeZ6DtEeSJP70eRk6eCg”
        base_InitialNode=”_kUIeZ6DtEeSJP70eRk6eCg”/>
```

W modelu BPEL po transformacji (Listing 3) elementem odpowiadającym za rozpoczęcie przepływu jest Receive z atrybutem createInstance. Wymagane jest również utworzenie zmiennej, do której zostaną przekazane dane wejściowe z metody uruchamiającej proces.

Listing 3. Fragmenty kodu BPEL po transformacji elementu rozpoczynającego przepływ mediacyjny „Wyzwalacz ESB”

```
<bpel:partnerLinks>
  <!-- Relacja z usługą sieciową, która jest interfejsem dla procesu -->
  <bpel:partnerLink name=”plESB”/>
  ...
</bpel:partnerLinks>
<bpel:variables>
  <!-- Zmienna z danymi wejściowymi -->
  <bpel:variable messageType=”createZlecenie” name=”CreateZlecenieIn”/>
  ...
</bpel:variables>
```

```
...
<!-- Utworzenie instancji procesu po wywołaniu metody usługi sieciowej -->
<bpel:receive createInstance="yes"
        name="createZlecenieStart"
        partnerLink="plESB"
        variable="CreateZlecenieIn"/>
```

c. *Wywołanie metody usługi sieciowej*

Wywołanie metody usługi sieciowej w modelu UML reprezentuje element OpaqueAction ze stereotypem Endpoint (Listing 4). Element ten powinien być powiązany relacją „Usage" z metodą w interfejsie, aby wskazać metodę usługi sieciowej, która ma zostać wywołana. Przykładowe kody zostały opisane dla „Znajdź zlecenie" (rysunek 4).

Listing 4. Fragmenty kodu UML elementu wywołującego metodę usługi sieciowej w kroku „Znajdź zlecenie"

```
<!-- Element UML wywołujący metodę usługi sieciowej -->
<node xmi:type="uml:OpaqueAction"
    xmi:id="_kUIecaDtEeSJP70eRk6eCg"
    name="Znajdź zlecenie"
    clientDependency="_kUN9S6DtEeSJP70eRk6eCg"
    outgoing="_kUIeg6DtEeSJP70eRk6eCg"
    incoming="_kUIegKDtEeSJP70eRk6eCg"
    inPartition="_kUIevKDtEeSJP70eRk6eCg">
...
<!-- Relacja pomiędzy Endpoint a metodą w usłudze sieciowej -->
<packagedElement xmi:type="uml:Usage"
        xmi:id="_kUN9S6DtEeSJP70eRk6eCg"
        name=""
        supplier="_kUIeR6DtEeSJP70eRk6eCg"
        client="_kUIecaDtEeSJP70eRk6eCg"/>
...
<packagedElement xmi:type="uml:Package"
        xmi:id="_kUIeRaDtEeSJP70eRk6eCg"
        name="Dom Maklerski">
  <packagedElement xmi:type="uml:Interface"
        xmi:id="_kUIeRqDtEeSJP70eRk6eCg"
        name="ZlecenieService"
```

```
                clientDependency="_kUIeVaDtEeSJP70eRk6eCg">
    <!-- Wywoływana metoda w usłudze sieciowej-->
    <ownedOperation xmi:id="_kUIeR6DtEeSJP70eRk6eCg" name="findAllZlecenie">
      <ownedParameter xmi:id="_kUIeSKDtEeSJP70eRk6eCg" name="arg0"/>
    </ownedOperation>
</packagedElement>
<!-- Stereotyp Endpoint dodany do OpaqueAction -->
<EIP:Endpoint xmi:id="_kUN9dqDtEeSJP70eRk6eCg"
          base_Action="_kUIecaDtEeSJP70eRk6eCg"
          base_ActivityNode="_kUIecaDtEeSJP70eRk6eCg"/>
```

W modelu BPEL po transformacji odpowiadającym elementem jest Invoke (Listing 5). Wymaga podania usługi sieciowej oraz wywoływanej operacji, zmiennej wejściowej, z której przekazane zostaną dane w argumentach operacji, oraz zmiennej wyjściowej, w której przechowywane będzie to, co zwróci wykonana operacja.

Listing 5. Fragmenty kodu BPEL elementu wywołującego metodę usługi sieciowej w kroku „Znajdź zlecenie"

```
<bpel:partnerLinks>
  <!-- Relacja z usługą sieciową-->
  <bpel:partnerLink name="plZlecenieDomMaklerski"
                partnerLinkType="ZlecenieDomMaklerskiWebServiceLinkType"
                partnerRole="ZlecenieDomMaklerskiWebServiceRole"/>
  ...
</bpel:partnerLinks>
...
<bpel:variables>
  <!-- Dane wejściowe dla metody -->
  <bpel:variable messageType="findAllZlecenie" name="FindAllZlecenieIn"/>
  <!-- Dane wyjściowe dla metody -->
  <bpel:variable messageType="findAllZlecenieResponse" name="FindAllZlecenieOut"/>
  ...
</bpel:variables>
...
<!-- Wywołanie metody w usłudze sieciowej -->
<bpel:invoke inputVariable="FindAllZlecenieIn"
          name="findAllZlecenie"
          operation="findAllZlecenie"
          outputVariable="FindAllZlecenieOut"
          partnerLink="plZlecenieDomMaklerski" />
```

d. Transformacja komunikatów

Przed wywołaniem operacji w usłudze sieciowej należy przygotować odpowiednie dane wejściowe. W BPEL można odczytywać dane z wyjść usług sieciowych, przekształcać dane oraz przypisać je do wejść usług sieciowych. W modelu UML wykorzystuje się do tego wzorce mediacyjne z kategorii transformacja, np. ContentEnricher, Translator. Transformację danych należy zdefiniować przy użyciu języka XPath oraz takich samych znaczników transformacji danych, z jakich korzysta się w BPEL. Poniżej zostały przedstawione fragmenty kodu UML (Listing 6) dla wzbogacania treści (ang. *ContentEnricher*) na podstawie „Ustaw parametr status OCZEKUJACE i limit" (rysunek 4). Przygotowany komunikat wysłany jest w kroku „Znajdź zlecenie". Dla danych wejściowych operacji findAllZlecenie przypisywane są dwa argumenty 'OCZEKUJACE' oraz '1'.

Listing 6. Fragmenty kodu UML wzorca ContentEnricher w kroku „Ustaw parametr status OCZEKUJACE i limit"

```
<node xmi:type="uml:OpaqueAction" xmi:id="_kUIeaKDtEeSJP70eRk6eCg" name="Ustaw
parametr status OCZEKUJACE i limit" outgoing="_kUIem6DtEeSJP70eRk6eCg" incoming="_kUI-
emKDtEeSJP70eRk6eCg" inPartition="_kUIevaDtEeSJP70eRk6eCg">
  <body>
    <copy>
      <from><![CDATA['OCZEKUJACE']]></from>
      <to><![CDATA[$FindAllZlecenieIn.parameters/arg0]]></to>
    </copy>
    <copy>
      <from><![CDATA[10]]></from>
      <to><![CDATA[$FindAllZlecenieIn.parameters/arg1]]></to>
    </bpel>
  </body>
  </node>
  ...
  <!-- Stereotyp ContentEnricher dodany do OpaqueAction -->
  <EIP:ContentEnricher xmi:id="_kUN9bqDtEeSJP70eRk6eCg" base_Action="_kUIeaKDtEeS-
JP70eRk6eCg"/>
```

Kod transformacji należy wpisać w atrybucie „body" wybranego elementu. W IBM RSA można to wykonać według poniższych kroków:

- na diagramie należy zaznaczyć odpowiedni element. Dla wybranego elementu zostaną wyświetlone ustawienia w oknie Properties;
- w oknie Properties należy wybrać zakładkę Advanced. Następnie na liście odnaleźć i kliknąć w Value dla atrybutu body, wyświetli się okno Body.
- W wyświetlonym oknie w polu Value wprowadzić kod, następnie kliknąć Add, potem OK.

Poniżej został przedstawiony wynik transformacji w Integration2BPEL dla wzbogacania treści (Listing 7).

Listing 7. Fragmenty kodu BPEL wzorca ContentEnricher w kroku „Ustaw parametr status OCZEKUJACE i limit"

```
<bpel:assign name="UstawParametrStatusOczekujaceILimit" validate="no">
 <bpel:copy>
  <bpel:from><![CDATA['OCZEKUJACE']]></bpel:from>
  <bpel:to><![CDATA[$FindAllZlecenieIn.parameters/arg0]]></bpel:to>
 </bpel:copy>
 <bpel:copy>
  <bpel:from><![CDATA[10]]></bpel:from>
  <bpel:to><![CDATA[$FindAllZlecenieIn.parameters/arg1]]></bpel:to>
 </bpel:copy>
</bpel:assign>
```

e. *Zakończenie przepływu integracyjnego*

W modelu UML ostatnim elementem diagramu kończącym przepływ jest ActivityFinalNode ze stereotypem Endpoint (Listing 8). ActivityFinalNode podobnie jak w przypadku pierwszego elementu InitialNode należy połączyć relacją typu Usage z metodą w interfejsie, która rozpoczyna przepływ. Potrzebne jest to, aby interpreter posiadał informacje, gdzie zwrócić dane wyjściowe po zakończeniu przepływu.

Listing 8. Wybrane fragmenty kodu UML elementu kończącego przepływ integracyjny

```
<!-- Element UML kończący przepływ integracyjny -->
<node xmi:type="uml:ActivityFinalNode"
    xmi:id="_kUIea6DtEeSJP70eRk6eCg"
    clientDependency="_kUN9XaDtEeSJP70eRk6eCg"
    incoming="_kUIeuaDtEeSJP70eRk6eCg"/>
...
<!-- Metoda w usłudze sieciowej rozpoczynają przepływ -->
<ownedOperation xmi:id="_kUN9WaDtEeSJP70eRk6eCg" name="createZlecenie"/>
...
<!-- Relacja pomiędzy Endpoint a metodą w usłudze sieciowej -->
<packagedElement xmi:type="uml:Usage"
            xmi:id="_kUN9XaDtEeSJP70eRk6eCg"
            name="" supplier="_kUN9WaDtEeSJP70eRk6eCg"
            client="_kUIea6DtEeSJP70eRk6eCg"/>
```

```
...
<!-- Stereotyp Endpoint dodany do Node -->
<EIP:Endpoint xmi:id="_kUN9dKDtEeSJP70eRk6eCg" base_ActivityNode="_kUIea6DtEeS-
JP70eRk6eCg"/>
```

W modelu BPEL po transformacji elementem końcowym jest Reply, który zwraca dane wyjściowe poprzez metodę createZlecenie.

Listing 9. Wybrane fragmenty kodu BPEL elementu kończącego przepływ integracyjny

```
<bpel:variables>
  <!-- Zmienna z danymi wyjściowymi zwracanymi po zakończeniu procesu -->
  <bpel:variable messageType="createZlecenieResponse" name="CreateZlecenieOut"/>
  ...
</bpel:variables>
...
<!-- Element BPEL kończący przypływ mediacyjny -->
<bpel:reply name="createZlecenieEnd"
        operation="createZlecenie"
        partnerLink="plESB"
        variable="CreateZlecenieOut"/>
```

f. Iteracja po modelu UML i mapowanie elementów UML na elementy BPEL

W kodzie BPEL istotna jest kolejność elementów, interpretator odczytuje kolejne polecenia i wykonuje je. W kodzie UML natomiast nie jest istotna kolejność zapisu elementów. Interpreter odczytuje ją z elementów Edge, korzystając z atrybutów id, source, target. Z przedstawionego fragmentu kodu (Listing 10) można odczytać, że po elemencie „Wyzwalacz ESB" (source="_kUIeZ6DtEeSJP70eRk6eCg") następuje „Ustaw parametr status OCZEKUJACE i limit" (target="_kUIebKDtEeS-JP70eRk6eCg"). Program transformacji UML2BPEL przechodzi według tej zasady po kolejnych elementach, dodając kolejno do modelu BPEL odpowiednie elementy zgodnie z tabelą mapowań (tabela 3).

Listing 10. Kolejność elementów UML

```
<node xmi:type="uml:InitialNode"
    xmi:id="_kUIeZ6DtEeSJP70eRk6eCg"
    name="Wyzwalacz ESB"
    clientDependency="_kUN9XKDtEeSJP70eRk6eCg"
```

```
        outgoing="_kUIelaDtEeSJP70eRk6eCg"/>
    <node xmi:type="uml:OpaqueAction"
        xmi:id="_kUIeaKDtEeSJP70eRk6eCg"
        name="Ustaw parametr status OCZEKUJACE i limit"
        outgoing="_kUIem6DtEeSJP70eRk6eCg"
        incoming="_kUIemKDtEeSJP70eRk6eCg"
        inPartition="_kUIevaDtEeSJP70eRk6eCg">
        ...
        </node>
    ...
    <edge xmi:type="uml:ControlFlow"
        xmi:id="_kUIelaDtEeSJP70eRk6eCg"
        source="_kUIeZ6DtEeSJP70eRk6eCg"
        target="_kUIebKDtEeSJP70eRk6eCg">
    </edge>
```

W tabeli 3 podsumowano z poprzednich punktów mapowanie elementów UML na elementy BPEL podlegające transformacji Integration2BPEL.

Tabela 3

Mapowanie elementów UML na elementy BPEL

Wzorzec mediacyjny	UML XPath	BPEL XPath	Opis transformacji UML2BPEL
Endpoint	packagedElement[@ xmi:type=”uml:Activity”] /Node[@ xmi:type=”uml:InitialNode”]	Sequence /Receive[@ createInstance=”yes”]	Pierwszy element diagramu aktywności InitialNode z Endpoint zostaje zamieniony na Receive, który nasłuchuje otrzymanie żądania.
Endpoint	packagedElement[@ xmi:type=”uml:Activity”] /Node[@ xmi:type=”uml:InitialNode”]	Variables/Variable	Dla InitialNode z Endpoint, który powiązany jest relacją Usage z operacją, tworzona jest zmienna będąca wiadomością przychodzącą o nazwie: nazwa operacji + „In”.
Endpoint	packagedElement[@ xmi:type=”uml:Activity”] /Node[@ xmi:type=”uml:OpaqueAction”]	Sequence/Invoke	Node z Endpoint zostaje zamieniony na Invoke wywołujący operacje z usługi sieciowej z relacji Usage.
Endpoint	packagedElement[@ xmi:type=”uml:Activity”] /Node[@ xmi:type=”uml:OpaqueAction”]	Variables/Variable	Dla Node z Endpoint, który powiązany jest relacją Usage z operacją, tworzone są dwie zmienne dla wiadomości nadawanej i przychodzącej o nazwach: • nazwa operacji + „In”, • nazwa operacji + „Out”.
Content Enricher	packagedElement[@ xmi:type=”uml:Activity”] /Node	Sequence/Assign	Elementy transformacji wiadomości zostają zamienione na elementy Assign. Kody XPath z atrybutu body elementu Node zostają skopiowane do Assign.
Translator	packagedElement[@ xmi:type=”uml:Activity”] /Node	Sequence/Assign	Elementy transformacji wiadomości zostają zamienione na elementy Assign. Kody XPath z atrybutu body elementu Node zostają skopiowane do Assign.
Content Based Router	packagedElement[@ xmi:type=”uml:Activity”] /Node	Sequence/If-else	Node z ContentBasedRouter zostaje zamieniony na If-else.
Endpoint	packagedElement[@ xmi:type=”uml:Activity”] /ActivityFinalNode	Sequence/Reply	Ostatni element aktywności ActivityFinalNode z Endpoint zostaje zamieniony na Reply, który jest portem wyjściowym z procesu.
Endpoint	packagedElement[@ xmi:type=”uml:Activity”] /ActivityFinalNode	Variables/Variable	Dla ActivityFinalNode z Endpoint, który powiązany jest relacją Usage z operacją, tworzona jest zmienna będąca wiadomością wychodzącą o nazwie: nazwa operacji + „Output”.

g. Model docelowy

W wyniku działania transformacji Integration2BPEL dla diagramu aktywności przedstawiającego przepływ integracyjny (rysunek 4) zostaje utworzony przepływ integracyjny w języku BPEL o nazwie takiej samej jak diagram aktywności UML. Przykład modelu wyjściowego w postaci kodu BPEL utworzonego w wyniku działania transformacji Integration2BPEL przedstawiono na listingu (Listing 11).

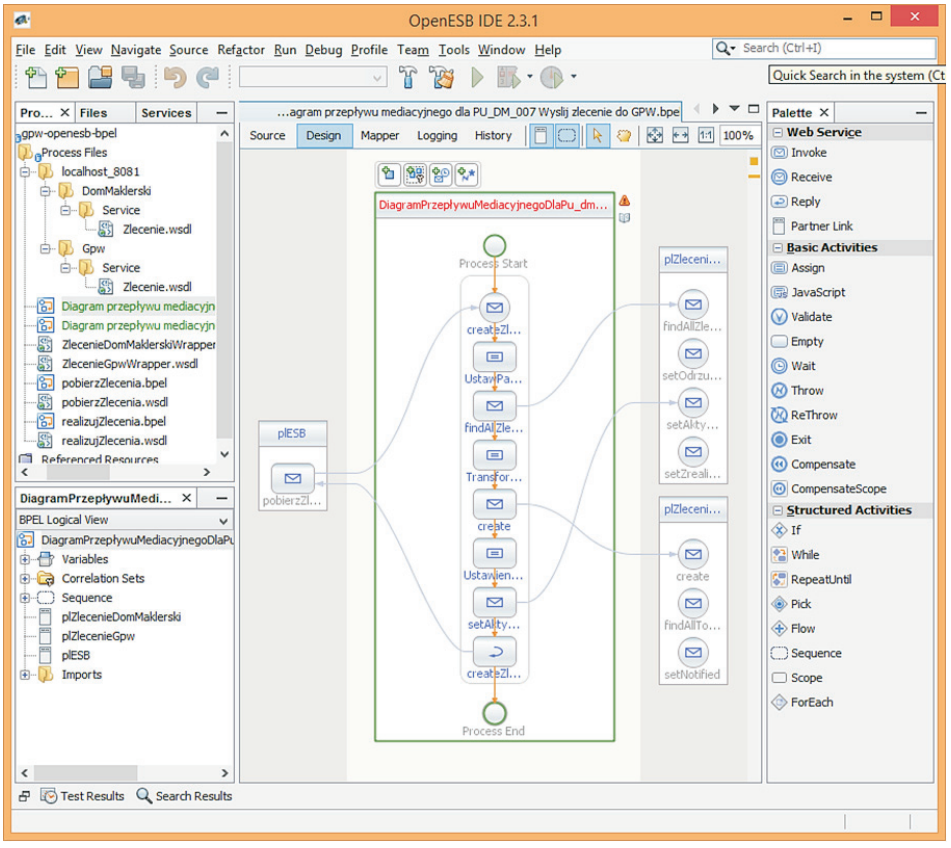
Listing 11. Wygenerowany kod BPEL

```
<?xml version="1.0" encoding="UTF-8"?>
<bpel:process exitOnStandardFault="yes"
    name="DiagramPrzepływuMediacyjnegoDlaPu_dm_007WyslijZlecenieDoGpw»
xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">
  <bpel:partnerLinks>
    <bpel:partnerLink name="plZlecenieDomMaklerski"
      partnerLinkType="ZlecenieDomMaklerskiWebServiceLinkType" partnerRole="Zlece
nieDomMaklerskiWebServiceRole"/>
    <bpel:partnerLink name="plZlecenieGpw"
      partnerLinkType="ZlecenieGpwWebServiceLinkType" partnerRole="ZlecenieGpw
WebServiceRole"/>
    <bpel:partnerLink name="plESB"/>
  </bpel:partnerLinks>
  <bpel:variables>
    <bpel:variable messageType="createZlecenie" name="CreateZlecenieIn"/>
    <bpel:variable messageType="findAllZlecenie" name="FindAllZlecenieIn"/>
    <bpel:variable messageType="findAllZlecenieResponse" name="FindAllZlecenieOut"/>
    <bpel:variable messageType="create" name="CreateIn"/>
    <bpel:variable messageType="createResponse" name="CreateOut"/>
    <bpel:variable messageType="setAktywne" name="SetAktywneIn"/>
    <bpel:variable messageType="setAktywneResponse" name="SetAktywneOut"/>
    <bpel:variable messageType="createZlecenieResponse" name="CreateZlecenieOut"/>
  </bpel:variables>
  <bpel:sequence>
    <bpel:receive createInstance="yes" name="createZlecenieStart"
      partnerLink="plESB" variable="CreateZlecenieIn"/>
    <bpel:assign name="UstawParametrStatusOczekujaceILimit" validate="no">
      <bpel:copy>
        <bpel:from><![CDATA['OCZEKUJACE']]></bpel:from>
        <bpel:to><![CDATA[$FindAllZlecenieIn.parameters/arg0]]></bpel:to>
      </bpel:copy>
```

```
    <bpel:copy>
      <bpel:from><![CDATA[10]]></bpel:from>
      <bpel:to><![CDATA[$FindAllZlecenieIn.parameters/arg1]]></bpel:to>
    </bpel:copy>
  </bpel:assign>
  <bpel:invoke inputVariable="FindAllZlecenieIn"
    name="findAllZlecenie" operation="findAllZlecenie"
    outputVariable="FindAllZlecenieOut" partnerLink="plZlecenieDomMaklerski"/>
  <bpel:assign name="TransformacjaDoFormatuZleceniaGpw" validate="no">
    <bpel:copy>
      <bpel:from><![CDATA[$FindAllZlecenieOut.parameters/return/id]]></bpel:from>
      <bpel:to><![CDATA[$CreateIn.parameters/arg0/id]]></bpel:to>
    </bpel:copy>
    <bpel:copy>
     <bpel:from><![CDATA[$FindAllZlecenieOut.parameters/return/ilosc]]></bpel:from>
      <bpel:to><![CDATA[$CreateIn.parameters/arg0/ilosc]]></bpel:to>
    </bpel:copy>
    <bpel:copy>
      <bpel:from><![CDATA[$FindAllZlecenieOut.parameters/return/ks]]></bpel:from>
      <bpel:to><![CDATA[$CreateIn.parameters/arg0/ks]]></bpel:to>
    </bpel:copy>
    <bpel:copy>
     <bpel:from><![CDATA[$FindAllZlecenieOut.parameters/return/limit]]></bpel:from>
      <bpel:to><![CDATA[$CreateIn.parameters/arg0/limit]]></bpel:to>
    </bpel:copy>
    <bpel:copy>
     <bpel:from><![CDATA[$FindAllZlecenieOut.parameters/return/spolka]]></bpel:from>
       <bpel:to><![CDATA[$CreateIn.parameters/arg0/spolka]]></bpel:to>
    </bpel:copy>
  </bpel:assign>
  <bpel:invoke inputVariable="CreateIn" name="create"
    operation="create" outputVariable="CreateOut" partnerLink="plZlecenieGpw"/>
  <bpel:assign name="UstawienieParametrówDoAktywacjiZlecenia» validate="no">
    <bpel:copy>
      <bpel:from><![CDATA[$FindAllZlecenieOut.parameters/return/id]]></bpel:from>
      <bpel:to><![CDATA[$SetAktywneIn.parameters/arg0]]></bpel:to>
    </bpel:copy>
  </bpel:assign>
  <bpel:invoke inputVariable="SetAktywneIn" name="setAktywne"
```

```
            operation="setAktywne" outputVariable="SetAktywneOut" partnerLink="plZlecenie
DomMaklerski"/>
          <bpel:reply name="createZlecenieEnd" operation="createZlecenie"
            partnerLink="plESB" variable="CreateZlecenieOut"/>
        </bpel:sequence>
      </bpel:process>
```

Wygenerowany kod BPEL może być wczytywany i interpretowany na magistrali usług OpenESB. Model BPEL należy uzupełnić o fizyczne adresy plików WSDL usług. Przykład kodu BPEL utworzonego w wyniku działania transformacji Integration2BPEL, otwarty w OpenESB, przedstawiono na rysunku 7.

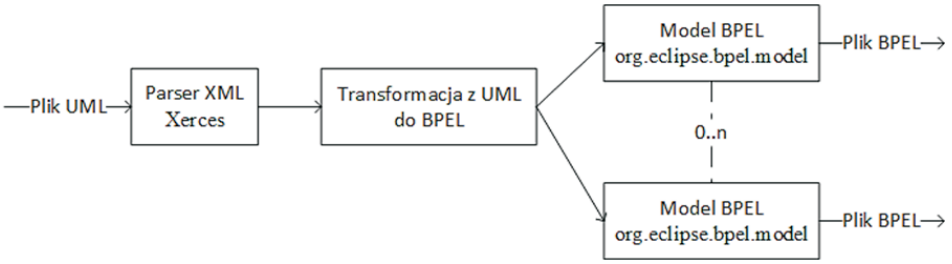


Rys. 7. Przepływ integracyjny BPEL w OpenESB dla *Utwórz zlecenie*

7. Implementacja transformacji Integration2BPEL

Istnieją dedykowane języki transformacji modeli, np. QVT, ATL. Zaletami ich stosowania jest mniejsza ilość kodu i większa jego przejrzystość. Wadami tych języków jest mniejsza popularność w stosunku do języków ogólnego przeznaczenia, takich jak Java. Przez to mniejsza jest ilość dostępnej dokumentacji i liczba przykładów. Zdobycie odpowiedniej wiedzy znacznie wydłuża proces przygotowania transformacji. Obecne implementacje języków transformacji nie są w pełni stabilne i podczas pracy można spotkać się z różnymi błędami, o których można szerzej przeczytać na forach dyskusyjnych. Ponadto, przy trudniejszych problemach kłopotliwe staje się używanie dedykowanych języków transformacji i należy stosować obejścia. Wyżej wymienione zagadnienia oraz potrzeba budowy elastycznej transformacji spowodowały wybór języka Java do jej implementacji. Uniwersalność języka Java pozwala na napisanie skomplikowanych transformacji w krótszym czasie. Wadami natomiast jest duża ilość kodu i słabsza czytelność. Skorzystanie z języka Java pozwoli w przyszłości na utworzenie pluginu do Eclipsa z transformacją Integration2BPEL.

Rysunek 8 przedstawia schemat przepływu sterowania i danych w transformacji Integration2BPEL.



Rys. 8. Schemat przypływu danych przez transformację Integration2BPEL

Implementacja transformacji Integration2BPEL została wykonana w postaci klas języka Java. Transformacja korzysta z następujących bibliotek: Xerces (parser do odczytywania XML), Eclipse Model Framework, org.elispse.bpel.model, org.eclipse.wst.wsdl. Pliki UML z modelem przepływu integracyjnego odczytywane są przez parser XML – Xerces. Elementy UML przekształcone zostają w strukturę obiektów z biblioteki org.eclipse.bpel.model i org.eclipse.wst.wsdl opisującą model BPEL. Utworzony w ten sposób wykonywalny przepływ integracyjny przy wykorzystaniu Eclipse Modeling Framework zostaje zapisany w pliku XML w formacie języka BPEL. Zaprojektowane zostały dwie klasy:

— UmlToBpel — odczytuje i parsuje plik UML i dla każdego diagramu aktywności z przepływem mediacyjnym tworzy nową instancję klasy CreateBpel, do której przekazuje wskaźnik na diagram;

— CreateBpel — obiekt tej klasy otrzymuje wskaźnik do konkretnego diagramu aktywności z modelu UML. Jeżeli jest prawidłowym diagramem aktywności z przepływem integracyjnym, zostaje utworzony plik BPEL o takiej samej nazwie jak diagram aktywności UML.

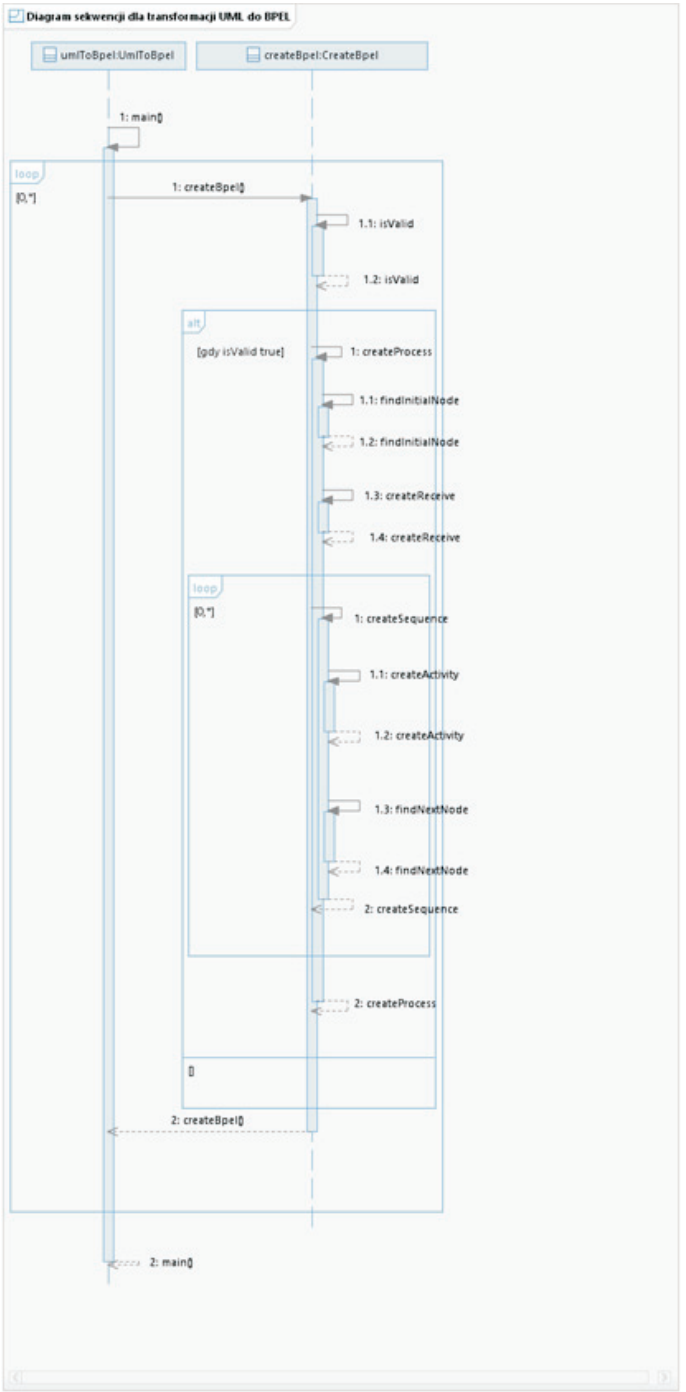
Tabela 4 przedstawia metody klasy CreateBPEL.

Tabela 4

Metody klasy CreateBPEL

Metoda	Opis metody
isValid	Sprawdza, czy diagram aktywności jest prawidłowym diagramem aktywności przedstawiającym przepływ integracyjny z wykorzystaniem profilu „UML Profile for Integration Flows".
initBpel	Tworzy nowy plik z rozszerzeniem .bpel o nazwie takiej samej jak przetwarzany diagram aktywności.
getEIPs	Pobiera kolekcję Map ze wszystkimi stereotypami przypisanymi do elementów UML.
createProcess	Tworzy proces BPEL.
createSequence	Metoda wywoływana rekurencyjnie przy każdym rozwidleniu przepływu na diagramie aktywności.
createReceive	Tworzy aktywność Receive, która jest pierwszym elementem BPEL.
createActivity	Tworzy aktywności: Invoke, Assign w zależności od Node z UML podanego jako parametr.
createDecision	Tworzy element BPEL IF-ELSE.
getNextNode	Dla wybranego Node odnajduje kolejne Node zgodnie z kierunkiem przepływu.
getPrevNode	Dla wybranego Node odnajduje poprzedni Node zgodnie z kierunkiem przepływu.
findOperationByR	Odnalezienie dla wybranego elementu relacji do operacji.
findNodeById	Odnalezienie Node z przypisanym atrybutem id podanym w parametrze metody.
findNodeByClient	Odnalezienie Node z przypisanym atrybutem client podanym w parametrze metody.
findInitialNode	Odnalezienie pierwszego Node inicjującego przepływ.

Rysunek 9 przedstawia diagram sekwencji dla działania transformacji Integration2BPEL.



Rys. 9. Diagram sekwencji działania transformacji Integration2BPEL

9. Podsumowanie

Zaprojektowana transformacja Integration2BPEL oraz zastosowanie podejścia MDD przynosi korzyści dla procesu projektowania platformy integracyjnej. Dzięki transformacji możliwe jest skrócenie czasu implementacji wykonywalnego przepływu integracyjnego przez usunięcie z procesu projektowania zadania przygotowania modelu BPEL. W przypadku zmiany w przebiegu przepływu integracyjnego możliwe jest ponowne wygenerowanie wykonywalnego przepływu integracyjnego BPEL. Stosowanie transformacji umożliwia łatwiejsze wprowadzanie zmian w przepływach integracyjnych. Zmiany te należy wprowadzać z poziomu modelu przepływu integracyjnego w UML (model PIM) i generować przepływ w BPEL (model PSM). Dzięki temu uzyskuje się spójność modelu z wykonywalnym przepływem integracyjnym.

W toku dalszych prac rozważane jest rozbudowanie transformacji Integration2BPEL w transformację dwukierunkową. Założeniem takiej modyfikacji byłaby możliwość aktualizacji modelu przepływu integracyjnego w UML zmianami dokonanymi w wykonywalnym przepływie integracyjnym w BPEL. Odrębnym zagadnieniem jest rozszerzenie transformacji o możliwość aktualizacji już raz wygenerowanego wykonywalnego przepływu w BPEL. Zmiany wprowadzone w PIM uzupełniałyby model PSM, nie usuwając wcześniej wprowadzonych zmian w modelu PSM. Celowe byłoby także utworzenie wtyczki Eclipse, która korzystałaby z transformacji Integration2BPEL i umożliwiała uruchomienie transformacji z menu kontekstowego w IBM Rational Software Architect. Planuje się także połączenie dwóch transformacji generujących przepływy integracyjne do języka Java oraz do BPEL z jednego modelu przepływu integracyjnego w UML. Możliwe będzie dzięki temu generowanie wykonywalnych przepływów integracyjnych w różnych językach i na różne platformy wykonawcze. W szczególności na platformę IBM Integration Bus, IBM WebSphere Application Server oraz OpenESB.

Źródło finansowania pracy — środki własne autorów.

Artykuł wpłynął do redakcji 12.03.2016 r. Zweryfikowaną wersję po recenzjach otrzymano 28.05.2018 r.

LITERATURA

[1] He W., Xu L.D., *Integration of Distributed Enterprise Applications: A Survey*, IEEE Transactions on Industrial Informatics, 2012.

[2] IBM Integration Bus, http://www.ibm.com/middleware/integration/en-us/enterprise-service-bus-esb.html (11 lutego 2016).

[3] TIBCO Business Works, http://www.tibco.com/products/automation/application-integration/activematrix-businessworks/enterprise-service-bus (11 lutego 2016).

[4] Oracle Service Bus, http://www.oracle.com/us/products/middleware/soa/service-bus/overview/index.html (11 lutego 2016).

[5] JBoss Fuse, http://www.jboss.org/products/fuse/overview/ (11 lutego 2016).

[6] Mule ESB, https://www.mulesoft.com/platform/soa/mule-esb-open-source-esb (11 lutego 2016).

[7] Petals ESB, http://petals.ow2.org (11 lutego 2016).

[8] WSO2 Enterprise Service Bus, http://wso2.com/products/enterprise-service-bus/ (11 lutego 2016).

[9] OpenESB, http://www.open-esb.net (11 lutego 2016).

[10] Martínez-Carreras M.A., García Jimenez F.J., Gómez Skarmeta A.F., *Building integrated business environments: analysing open-source ESB*, Enterprise Information Systems, 9, 2015, 401-435.

[11] Vollmer K., Gilpin M., Sander R., *The Forrester WaveTM: Enterprise Service Bus*, Q2. Forrester Research Report, 2011.

[12] Umar A., Zordan A., *Reengineering for Service Oriented Architectures: A Strategic Decision Model for Integration Versus Migration*, Journal of Systems and Software, 82, 3, 2008, 448–462.

[13] Wieczorkowski J., *Zastosowanie oprogramowania standardowego w administracji publicznej*, Roczniki Kolegium Analiz Ekonomicznych, 29, 2013, 343-352.

[14] Unified Modeling Language Specification Version 2.4.1, OMG 2011, http://www.omg.org /spec/UML/2.4.1/ (11 lutego 2016).

[15] Czarnecki A., *Wybrane metody i narzędzia modelowania systemów informatycznych z użyciem języka UML*, [w:] Zarządzanie technologiami informatycznymi: przykłady zastosowań IT, red. Orłowski C., Pomorskie Wydawnictwo Naukowo-Techniczne, 2007, 51-59.

[16] Web Services Business Process Execution Language Version 2.0, OASIS Standard, 11 April 2007 (11 lutego 2016).

[17] Górski T., *UML profiles for architecture description of an integration platform*, Biuletyn Wojskowej Akademii Technicznej/Bulletin of Military University of Technology, 62, 2, 2013, 43-56.

[18] Górski T., A*rchitectural view model for an integration platform*, Journal of Theoretical and Applied Computer Science, 6, 1, 2012, 25–34.

[19] Hohpe G., Woolf B., *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison–Wesley, Boston 2003.

[20] Mens T., van Gorp P., *A taxonomy of model transformation*, Electronic Notes In Theoretical Computer Science, 152, 2006, 125–142.

[21] Papazoglou M.P. et al., *Service-oriented computing: State of the art and research challenges*, IEEE Computer, Nov., 2007, 38–45.

[22] Górski T., *Platformy integracyjne. Zagadnienia wybrane*, Wydawnictwo Naukowe PWN, Warszawa, 2012.

[23] Kleppe A.J., Warmer J., Bast W., *MDA Explained, The Model Driven Architecture: Practice and Promise*, Addison–Wesley, Boston, 2003.

[24] Ameller D., Burgués X., Collell O., Costal D., Franch X., Papazoglou M.P., *Development of service-oriented architectures using model-driven development: A mapping study*, Information and Software Technology, 62, 2015, 42–66.

[25] Yu J., Sheng Q.Z., Swee J.K.Y., Han J., Liu Ch., Noor T.H., *Model-driven development of adaptive web service processes with aspects and rules*, Journal of Computer and System Sciences, 81, 2015, 533–552.

[26] Khadka R., Sapkota B., Pires L.F., van Sinderen M., Jansen S., *Model-driven approach to enterprise interoperability at the technical service level, Computers in Industry*, 64, 2013, 951–965.

[27] Santiago I., Jiménez Á., Vara J.M., De Castro V., Bollati V.A., Marcos E., *Model-Driven Engineering as a new landscape for traceability management: A systematic literature review*, Information and Software Technology, 54, 2012, 1340–1356.

[28] Mazanek S., Hanus M., *Constructing a bidirectional transformation between BPMN and BPEL with a functional logic programming language*, Journal of Visual Languages and Computing, 22, 2011, 66–89.

[29] Cibrán M.A., *Translating BPMN Models into UML Activities*, Lecture Notes in Business Information Processing, 17, 2009, 236–247.

[30] Macek O., Richta K., *The BPM to UML activity diagram transformation using XSLT*, DATESO, 2009.

[31] Pattathe M., *Transformation to SOA: Part 4. How Web service processes transform from UML to BPEL in IBM Rational Software Architect*, 2008, http://www.ibm.com/developerworks/rational/library/08/0318_pattathe/ (11 lutego 2016).

[32] Zhang M., Duan Z., *From Business Process Models to Web Services Orchestration: The Case of UML 2.0 Activity Diagram to BPEL*, Conference: Service-Oriented Computing – ICSOC 2008, 6th International Conference, Sydney, Australia, 2008.

[33] Gorelik D., *Transformation to SOA: Part 3. UML to SOA*, 2008, http://www.ibm.com/developerworks/rational/library/08/0115_gorelik/ (11 lutego 2016).

[34] Bliźniuk G., Gzik T., Koszela J., *Translacja opisów ścieżek klinicznych z postaci GLIF na XPDL zapewniająca interoperacyjność z systemem EHR*, Biuletyn Instytutu Systemów Informatycznych, 9, 2012, 1-8.

[35] Górski T., *Model-to-model transformations of architecture descriptions of an integration platform*, Journal of Theoretical and Applied Computer Science, 8. 2, 2014, 48-62.

[36] Górski T., *Automatyzacja projektowania przepływów mediacyjnych*, [w:] Projektowanie systemów informatycznych: modele i metody, Wojskowa Akademia Techniczna, 2014, 19-34.

[37] Nowicki T., *Modelowanie, symulacja i analiza systemów w środowisku WebSphere Business Modeler*, Symulacja w Badaniach i Rozwoju, 2, 1, 2011, 23-36.

[38] OASIS Web Services Business Process Execution Language (WSBPEL), https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel (4 kwietnia 2017).

[39] Organization for the Advancement of Structured Information Standard, https://www.oasis-open.org/ (4 kwietnia 2017).

[40] Extensible Markup Language (XML), https://www.w3.org/XML/ (4 kwietnia 2017).

[41] Web Services Description Language (WSDL) 1.1, https://www.w3.org/TR/wsdl (4 kwietnia 2017).

[42] XML Path Language (XPath) 2.0 (Second Edition), https://www.w3.org/TR/xpath20/ (4 kwietnia 2017).

[43] SoaML, http://www.omg.org/spec/SoaML/

TOMASZ GÓRSKI, GRZEGORZ ZIEMSKI

UML activity diagram transformation into BPEL integration flow

Abstract. The growing interest of companies in integration and interoperability between information systems has caused increase in significance of Service-Oriented Architecture which provides tools for Enterprise Application Integration. In that architecture, Enterprise Service Bus provides technical possibilities of communication between IT systems. A key element in the communication are integration flows.

Objective: The aim of this article is to present a new transformation Integration2BPEL, which automates the development of executable integration flow expressed in the Web Services Business Process Execution Language (WS-BPEL) based on the model of the integration flow presented in the Unified Modelling Language (UML) activity diagram.

Method: The author proposes a transformation of the type of model-to-code type which generates integration flow expressed in WS-BPEL, which can be executed in any BPEL-compliant process engine. The integration flow is modelled using UML activity diagram with stereotypes from 'UML Profile for Integration Flows' profile in an IBM Rational Software Architect (RSA). Using Integration2BPEL transformation a complete, executable integration flow is generated, which is composed of many mediation mechanisms. Generated integration flows have been executed on OpenESB.

Results: The ability to generate a complete integration flow in BPEL, which without any additions can be run on enterprise service bus. Implementation phase of an integration flow construction was automated. Each of integration flows is implemented according to the same rules. In addition, it allows to avoid mistakes made by designers and programmers.

Conclusions: Model-Driven Development is an approach that leads to the automation of the design and programming phases. Integration2BPEL transformation is a uniform mechanism to design integration flow. Potentially, it also allows to avoid implementation errors.

Keywords: Web Services Business Process Execution Language (BPEL), Enterprise Service Bus (ESB), Unified Modelling Language (UML), UML activity diagram, Model-Driven Development (MDD), Transformation

DOI: 10.5604/01.3001.0012.6587