Biuletyn WAT
Vol. LXII, Nr 2, 2013

Transformacje do automatyzacji projektowania architektury platformy integracyjnej

TOMASZ GÓRSKI

Wojskowa Akademia Techniczna, Wydział Cybernetyki,
Instytut Systemów Informatycznych, 00-908 Warszawa, ul. gen. S. Kaliskiego 2,
gorski@wat.edu.pl, tomasz.gorski@rightsolution.pl

Streszczenie. W artykule przedstawiono transformacje automatyzujące projektowanie architektury platformy integracyjnej. Przyjęto model widoków architektonicznych „1 + 5" do przedstawiania architektury platformy integracyjnej. Do automatyzacji projektowania architektury platformy integracyjnej zastosowano transformacje typu model w model. Artykuł zawiera wprowadzenie do transformacji oraz przegląd aktualnej literatury. Pokazane zostały transformacje między modelami w widokach architektonicznych modelu „1 + 5": Integrowanych procesów, Przypadków użycia, Logiki oraz Integrowanych usług. Projekt i implementacja transformacji wykonane zostały w środowisku IBM Rational Software Architect. W podsumowaniu przedstawiono korzyści stosowania automatyzacji projektowania architektury systemów informatycznych oraz kierunki dalszych prac.
Słowa kluczowe: integracja systemów informatycznych, transformacje, inżynieria sterowana modelami

1. Wprowadzenie

Kluczowa dla firm jest jak najszybsza możliwość dostarczania usług lub towarów dostosowanych do potrzeb potencjalnych klientów. Kolejną istotną kwestią jest szybkość realizacji napływających zamówień lub obsługi wpływających spraw. Aby temu sprostać, należy zapewnić odpowiednie wsparcie dla procesów biznesowych przez systemy informatyczne eksploatowane w organizacji. Najczęściej eksploatowanych jest wiele systemów informatycznych w organizacji. Wymusza to potrzebę budowania rozwiązań integracyjnych składających się z systemów informatycznych oraz warstwy komunikacyjnej umożliwiającej im współpracę. Tego typu rozwiązanie nazywane jest platformą integracyjną lub rozwiązaniem integracyjnym. Przy projektowaniu

rozwiązań integracyjnych istotna jest możliwość modelowania ich kompletnego opisu architektonicznego. Do tego celu potrzebny jest model widoków architektonicznych dostosowany do potrzeb modelowania platform integracyjnych oraz zestaw konstrukcji modelowych, dzięki któremu można przedstawić pełną architekturę platformy integracyjnej [14, 17, 18]. Integracja wielu systemów informatycznych powoduje wysoką złożoność projektu integracji. Architekt Integracji [18], stosując transformacje, automatyzuje tworzenie modeli architektonicznych i dzięki temu skraca czas projektowania oraz nakład prac z tym związanych, a także zmniejsza liczbę błędów [19, 25, 35]. Szersza analiza tego zagadnienia została zamieszczona w sekcji 12 artykułu. Zastosowanie transformacji jest elementem podejścia inżynierii sterowanej modelami (ang. *Model-Driven Engineering* — MDE) [24, 26, 32].

Celem tego artykułu jest przedstawienie transformacji typu model w model automatyzujących projektowanie architektury platformy integracyjnej. Artykuł wykorzystuje model widoków architektonicznych „1 + 5" przedstawiony w [14]. Transformowane modele są na różnych poziomach abstrakcji. Z punktu widzenia notacji, trzem z prezentowanych transformacji podlegają modele wyrażone w tej samej notacji, języku UML, a jednej z nich podlegają modele wyrażone w różnych notacjach: BPMN [5, 8] oraz UML [34].

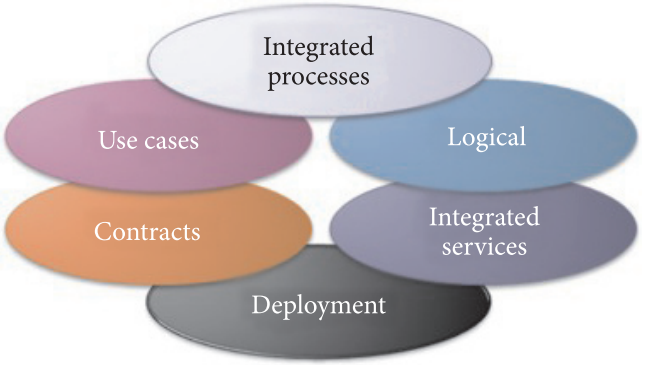
Pozostała część artykułu została ułożona w przedstawiony dalej sposób. W sekcji 2 przedstawiono widoki, modele i diagramy występujące w modelu widoków architektonicznych „1 + 5". W sekcji 3 przedstawiono wprowadzenie do transformacji. Sekcja 4 stanowi przegląd prac dotykających podobnych zagadnień. W sekcji 5 przedstawiono środowisko implementacji transformacji. Sekcja 6 to transformacje zaprojektowane dla automatyzacji projektowania architektury platformy integracyjnej. W sekcjach 7, 8, 9 i 10 przedstawiono szczegółowo cztery zaprojektowane transformacje. W sekcji 11 omówiono automatyzację tworzenia struktur modeli wynikowych transformacji. Sekcja 12 przedstawia podsumowanie i korzyści płynące z zastosowania transformacji w kontekście nakładu pracy, czasu realizacji projektu oraz liczby błędów. W sekcji tej przedstawiono także kierunki dalszych prac.

2. Model widoków architektonicznych „1 + 5"

Spójność opisu architektonicznego rozwiązań informatycznych jest istotnym zagadnieniem podlegającym aktualnym pracom badawczym [1]. Istnieje wiele modeli z różnymi zestawami widoków architektonicznych, np.: „4 + 1", RM-ODP, model Siemens, widoki SEI [31]. Natomiast nie zapewniają one możliwości kompletnego opisu architektury rozwiązań integracyjnych. Zaproponowano model widoków architektonicznych „1 + 5" dostosowany do potrzeb projektowania platform integracyjnych [14, 16]. W modelu tym wyodrębniono następujące widoki architektoniczne:

— Integrowanych procesów (ang. *Integrated processes*),
— Przypadków użycia (ang. *Use cases*),
— Logiki (ang. *Logical*),
— Integrowanych usług (ang. *Integrated services*),
— Kontraktów (ang. *Contracts*),
— Rozlokowania (ang. *Deployment*).

Podstawowym widokiem architektonicznym jest widok integrowanych procesów. W widoku tym modelowane są procesy biznesowe mające być zautomatyzowane na platformie integracyjnej. Następne cztery widoki (Przypadków użycia, Logiki, Integrowanych usług, Kontraktów) służą do przedstawienia projektu platformy integracyjnej. Widok *Przypadków użycia* zawiera wymagania funkcjonalne na system integrowany w ramach platformy integracyjnej. W widoku *Integrowanych usług* prezentowane są usługi wystawiane z systemów informatycznych oraz ich sposób włączenia na magistralę usług. Widok *Kontraktów* przedstawia komponenty reprezentujące systemy informatyczne i kontrakty określone między nimi. W widoku tym przedstawiane są także przepływy mediacyjne dla każdego kontraktu. Ostatni widok *Rozlokowania* przedstawia sposób osadzenia zaprojektowanych elementów platformy integracyjnej na określonym środowisku uruchomieniowym. Rysunek 1 przedstawia model widoków architektonicznych „1 + 5".



Rys. 1. Model widoków architektonicznych „1 + 5"

Szczegółowy opis przedstawianego modelu widoków architektonicznych, wraz z przykładami zastosowań, znajduje się w literaturze [14, 15, 16, 18]. W rozpatrywanym podejściu zaproponowane zostały modele oraz diagramy języków BPMN [5, 8] oraz UML [34] z rozszerzeniem o konstrukcje języka SoaML [33] do modelowania architektury platformy integracyjnej (tab. 1).

Tabela 1

Elementy do modelowania architektury platformy integracyjnej

Model	Widok	Diagram
Procesów	Integrowanych procesów	(BPMN) Procesu biznesowego
Przypadków użycia	Przypadków użycia	(UML) Przypadków użycia
		(UML) Aktywności
Projektowy	Logiki	(UML) Sekwencji
		(UML) Komunikacji
		(UML) Klas
Usług	Integrowanych usług	(UML) Komponentów
	Kontraktów	(UML) Komponentów
		(UML) Aktywności
		(UML) Struktury
Rozlokowania	Rozlokowania	(UML) Wdrożenia

3. Transformacje

W literaturze przedmiotu [7, 29] przyjmuje się następującą definicję transformacji typu model w model [24]: „Transformacja jest automatyczną generacją modelu docelowego z modelu źródłowego zgodnie z definicją transformacji. Definicja transformacji jest zbiorem reguł transformacji, które razem opisują, jak model w języku źródłowym może być transformowany w model w języku docelowym. Reguła transformacji jest opisem, jak jedna lub wiele konstrukcji z języka źródłowego może być transformowana w jedną lub wiele konstrukcji w języku docelowym". Po to aby można byłoby transformować modele, muszą być one wyrażone w jakimś języku modelowania. Każdy model musi być zgodny z metamodelem, który określa składnię i semantykę określonego typu modeli. W ten sam sposób metamodel musi być zgodny z metametamodelem. Metametamodel zwykle definiuje sam siebie, co oznacza, że może być wyspecyfikowany środkami własnej semantyki. Metamodele są zwykle definiowane przy użyciu diagramu klas języka UML. Natomiast istnieją także inne języki stosowane do tego celu [7], np. MetaObject Facility, Ecore metametamodel definiowany dla środowiska Eclipse Modeling Framework i Kernel MetaMetaModel. Ponadto stosowane są różne podejścia do definiowania i uruchamiania transformacji. Stosowane są podejścia bazujące na bezpośredniej manipulacji (ang. *direct-manipulation*), gdzie transformacje tworzone są w języku programowania operującym na modelach w pamięci operacyjnej komputera (np. Java Metadata Interface), deklaratywne, gdzie definiowane są reguły transformacji jako reguły matematyczne (np. Query/View/Transformation [30]) czy wykorzystujące

transformacje grafów, gdzie modele traktowane są jako grafy (np. *Attribute Graph Grammar*). Bazując na języku modelowania, w którym wyrażone są modele źródłowy i docelowy, można wyróżnić transformacje endogenne i egzogenne [29]. Transformacje endogenne dotyczą modeli wyrażonych w tym samym języku modelowania. Natomiast transformacje egzogenne dotyczą modeli wyrażonych w różnych językach modelowania. Transformacje można podzielić także na horyzontalne i wertykalne [29]. Z transformacją horyzontalną mamy do czynienia wtedy, gdy zarówno model źródłowy jak i docelowy są na tym samym poziomie abstrakcji. Natomiast transformację uznaje się za wertykalną, jeśli modele źródłowy i docelowy są na różnych poziomach abstrakcji. Ponadto transformacje modeli mogą być jednokierunkowe albo dwukierunkowe [29]. Jednokierunkowe umożliwiają wygenerowanie modelu docelowego z modelu źródłowego, a dwukierunkowe umożliwiają utrzymanie spójności między modelami źródłowym i docelowym niezależnie od tego, w którym z nich zostaną wprowadzone zmiany. W transformacji dwukierunkowej niezbędne jest zastosowanie związku śledzenia (ang. *traceability*) umożliwiającego powiązanie elementów między transformowanymi modelami [32].

4. Prace powiązane tematycznie

Przegląd literatury dokonany został zgodnie z procesem opisanym w metodzie Systematic Review [23]. Literatura przedmiotu jest bogata w zagadnienia transformacji modeli. Szczegółowa taksonomia transformacji modeli przedstawiona została w [29]. W literaturze opisywane są różne języki tworzenia transformacji: Query/View/Transformation (QVT) [30], ATL (Atlas Transformation Language) [3, 21]. Szczególnie aktualnymi i istotnymi zagadnieniami w publikacjach w ostatnich latach są weryfikacja i walidacja transformacji typu model w model. Weryfikacja i walidacja transformacji modeli z zastosowaniem niezmienników (ang. *invariants*) przedstawiona została w [6]. Ponadto przedstawiane są propozycje zastosowania języka QVT do budowy języka deklaratywnego do specyfikacji kontraktów wizualnych (ang. *visual contracts*) umożliwiającego weryfikację transformacji zdefiniowanych w dowolnym języku transformacji [20]. Aktualny, szeroki przegląd zagadnienia weryfikacji transformacji modeli znajduje się w [7].

W artykule skupiono się na przedstawieniu niezbędnych transformacji modeli do automatyzacji projektowania określonego typu systemów, jakimi są platformy integracyjne. Transformacje te zostały zaprojektowane zgodnie z wymogami modelu widoków architektonicznych „1 + 5". Zaproponowana transformacja BPMN do diagramu przypadków użycia jest nową, autorską propozycją identyfikacji przypadków użycia systemów informatycznych podlegających integracji. Zastosowano w niej także nowy, autorski profil „UML Profile for Integration Platform" m.in. ze stereotypem <<IntegratedSystem>> [17]. Zagadnienie transformacji modeli wyrażonych BPMN

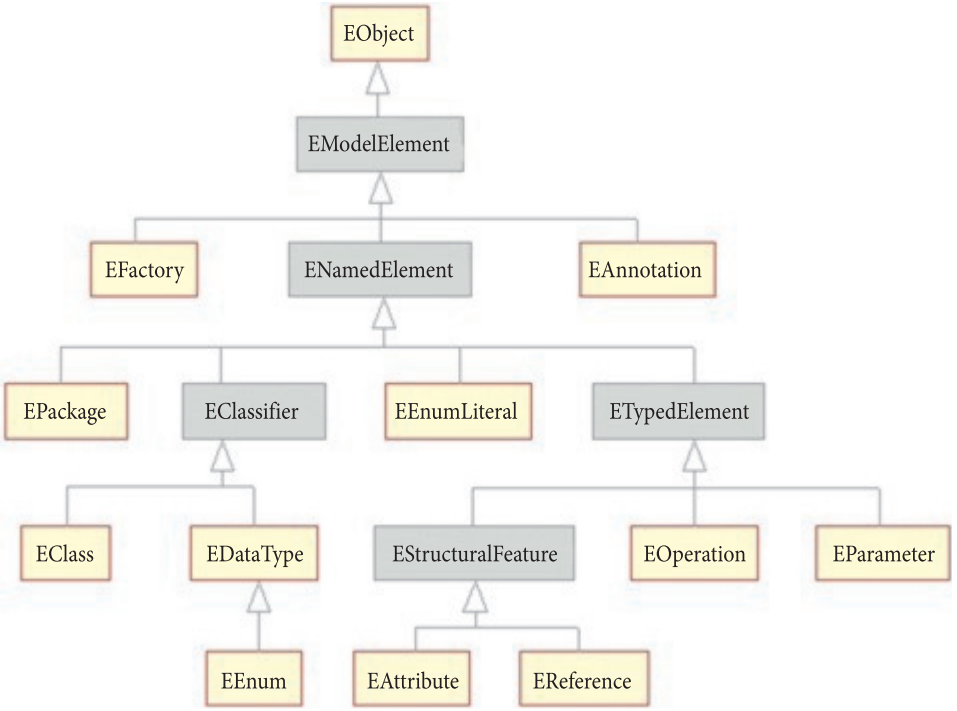
jest aktualne w literaturze. W [28] przedstawiono zagadnienie transformacji dwukierunkowej między modelami BPMN i BPEL. Dostępne są także prace pokazujące transformacje modeli przekształcające modele BPMN w modele UML. Natomiast dotyczą one transformacji BPMN w diagramy aktywności języka UML [10, 27]. W pozycji [13] przedstawiono podejście do dynamicznej identyfikacji usług z modelu BPMN, ale jawnie nie są tworzone modele w UML. W pracy tej zaproponowano metamodel do formalnej specyfikacji wymagań funkcjonalnych dla dynamicznej selekcji usług w modelach procesów biznesowych. Pozostałe, przedstawione w artykule transformacje modeli dotyczą transformowania modeli UML. Dwie z nich są nowymi, autorskimi propozycjami i dotyczą transformacji modelu *Przypadków użycia* do modelu *Usług* zaproponowanego w modelu widoków architektonicznych „1 + 5". Ostatnia dotyczy transformacji modelu *Przypadków użycia* do modelu *Projektowego* i stanowi rozszerzenie transformacji, która dostępna była w narzędziu IBM Rational Software Architect w wersji 6.0 i tworzyła realizacje przypadków użycia i klasy analityczne ze stereotypami <<boundary>> oraz <<control>>.

5. Środowisko implementacji transformacji

W środowisku IBM Rational Software Architect (RSA) można używać dostępnych transformacji, a także budować własne transformacje. RSA umożliwia tworzenie następujących typów transformacji: kodu w model (generowane są modele na podstawie kodu źródłowego), modelu w kod (generowany jest kod źródłowy, np.: C++, Java, na podstawie modelu), model w model (generowany jest model, np. UML, BPMN, na podstawie innego modelu). W artykule rozpatrywane są transformacje typu model w model. Narzędzie RSA jest rozszerzeniem środowiska Eclipse i wykorzystuje Ecore metametamodel [11]. Rysunek 2 przedstawia hierarchię komponentów Ecore.

Każda transformacja składa się z trzech elementów: modelu źródłowego, modelu docelowego i mapowania pomiędzy nimi. Źródłem transformacji modelu w model jest zawsze element, który może być mapowany na obiekt klasy *Ecore model* lub profil UML. Obiekty klas *Package* i *Model* z pakietu *org.eclipse.uml2.uml* oraz *Definitions* z pakietu *com.ibm.xtools.bpmn2* mogą być wskazane jako źródło transformacji. Obiekty takich klas mogą być wskazane jako źródło transformacji. W Ecore określono definicje elementów modeli BPMN oraz UML. Plik *plugin/com.ibm.xtools.bpmn2/model/bpmn2.ecore* zawiera definicję modelu Ecore dla BPMN. Natomiast definicja modelu Ecore dla UML zawarta jest w pliku *plugin/org.eclipse.uml2.uml/model/UML.ecore*. Jako model docelowy transformacji należy również podać obiekt dziedziczący z *Ecore model* lub profil UML.

Ostatnim elementem definiującym jest mapowanie pomiędzy modelem źródłowym i docelowym. Mapowanie jest wyrażane na kilku poziomach. Na najwyższym



Rys. 2. Hierarchia komponentów Ecore (źródło: [11])

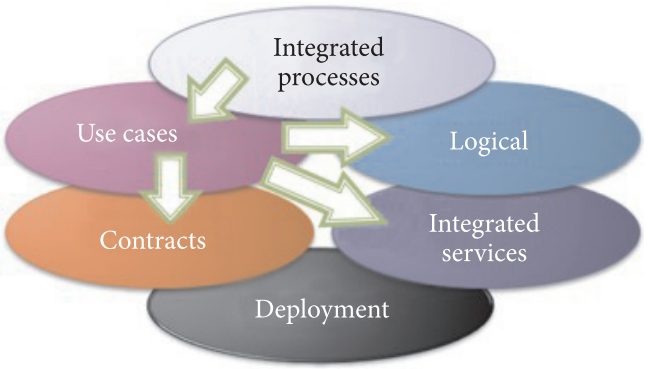
poziomie występuje mapowanie modelu źródłowego do modelu docelowego całej transformacji. Najwyższy poziom pokazuje główne elementy mapowania, np. *Definitions* w modelu źródłowym i *Package* w modelu docelowym. Następnie mogą być definiowane mapowania pomiędzy atrybutami zawartymi w elementach mapowania. Dla przykładu, atrybut *Name* z elementu *Definitions* może być transformowany w atrybut Name elementu *Package*. Przy definiowaniu mapowania pomiędzy atrybutami elementów należy określić typ mapowania. Dużą elastyczność budowy transformacji dają mapowania typu *Custom* i *Submap*. Zastosowanie mapowania typu *Custom* pozwala na podłączenie metody napisanej w języku Java, która wykona transformację pomiędzy atrybutami elementów. Zastosowanie mapowania typu *Submap* umożliwia dekompozycję tego mapowania i utworzenie wewnątrz niego mapowania zagnieżdżonego. Działanie transformacji implementowane jest w klasie Javy i wykonywane przez uruchamianie odpowiednich metod tej klasy. Metody klasy Java, realizujące transformację, mają dostępne modele źródłowy i docelowy, które zapisane są w postaci plików XML. Przykłady mapowania oraz metod klas języka Java implementujących transformacje przedstawione zostały dalej przy opisywaniu konkretnych transformacji.

6. Transformacje automatyzujące projektowanie architektury platformy integracyjnej

W pracach analitycznych i projektowych kluczowymi rolami są: *Główny Analityk* oraz *Główny Architekt*. W trakcie modelowania architektury platformy integracyjnej budowane są modele wymienione w tabeli (tab. 1). Prezentują one różne widoki architektoniczne. Pierwszy model, który jest tworzony, to model *Procesów*. Utworzenie tego modelu leży w odpowiedzialności *Głównego Analityka*. Mając utworzony model *Procesów*, należy pomyśleć o automatyzacji tworzenia modelu *Przypadków użycia*, który czerpie z tego pierwszego konstrukcje modelowe. Dzięki zastosowaniu transformacji możemy uzyskać część struktur i elementów modelu *Przypadków użycia* w sposób automatyczny. Następnie model *Przypadków użycia* musi zostać uzupełniony przez analityków. Dopiero do kompletnie wykonanego modelu *Przypadków użycia* możemy zastosować transformacje umożliwiające uzyskanie struktur i elementów modelowych w modelach *Projektowym* oraz *Usług*. W tych dwóch ostatnich modelach znajdują się elementy modelowe opisujące architekturę platformy integracyjnej w widokach *Logiki*, *Kontraktów* oraz *Integrowanych usług*. Zaproponowano tutaj trzy transformacje generujące elementy projektowe z modelu *Przypadków użycia* opisującego specyfikację wymagań funkcjonalnych. Model *Rozlokowania* opisuje fizyczne rozlokowanie oprogramowania na serwerach sprzętowych oraz środowiska uruchomieniowe. Został on wyłączony z obszaru automatyzacji tworzenia opisu architektonicznego platformy integracyjnej.

W związku z powyższym zaproponowano następujące transformacje (rys. 3):

— widoku integrowanych procesów do widoku przypadków użycia (BPM-N2UC),
— widoku przypadków użycia do widoku logiki (UC2Logical),
— widoku przypadków użycia do widoku integrowanych usług (UC2IS),
— widoku przypadków użycia do widoku kontraktów (UC2Contracts).



Rys. 3. Transformacje między widokami modelu „1 + 5”

Wszystkie zaprojektowane transformacje są wertykalne, co oznacza, że modele przez nie transformowane są na różnych poziomach abstrakcji. Ponadto, transformacja BPMN2UC jest egzogenna, gdyż podlegają jej modele wyrażone w różnych notacjach: BPMN oraz UML. Pozostałe rozpatrywane w artykule transformacje są endogenne i dotyczą modeli wyrażonych w języku UML. Pełna klasyfikacja transformacji przedstawiona została w [29].

7. Transformacja widoku *Integrowanych procesów* do widoku *Przypadków użycia*

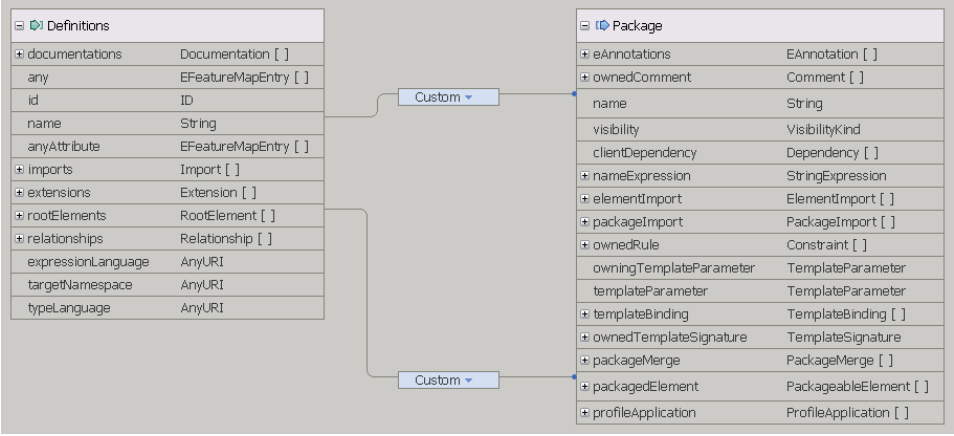
Celem transformacji jest uzyskanie modeli *Przypadków użycia* dla systemów informatycznych podlegających integracji. W widoku *Integrowanych procesów* przedstawiane są procesy, jakie zachodzą w organizacji. Przyjęto, że dla każdej organizacji występującej w modelu *Procesów* tworzony jest odrębny model *Przypadków użycia* określający specyfikację wymagań funkcjonalnych na odrębny system informatyczny. Na podstawie pracowników wyróżnionych w określonej organizacji tworzeni są aktorzy w ramach modelu *Przypadków użycia*, który odpowiada systemowi informatycznemu tworzonemu dla tej organizacji. Analogicznie zostało to przyjęte dla zadań w procesie biznesowym. Mianowicie, na podstawie zadań wyróżnionych w procesie określonej organizacji tworzone są przypadki użycia w ramach modelu *Przypadków użycia*, który odpowiada systemowi informatycznemu tworzonemu dla tej organizacji.

Na wejściu transformacji podawany jest model BPMN reprezentowany w RSA jako plik z rozszerzeniem *bpmx*. Transformacji mogą podlegać wszystkie typy diagramów BPMN stosowane w RSA: choreografii (ang. *choreography*), kolaboracji (ang. *collaboration*) i procesu (ang. *process*). Natomiast, ze względu na semantykę, powinna ona być stosowana do diagramów typu kolaboracji i procesu. Na model źródłowy wyrażony w BPMN nałożono warunki wstępne, po to aby transformacja mogła się wykonać poprawnie:

- każdy element typu zasób (ang. *Resource*) powinien reprezentować organizację,
- każdy element typu tor (ang. *Lane*) reprezentuje pracownika,
- każdy element typu tor reprezentujący pracownika powinien być powiązany z organizacją, do której należy przez element *PartitionElement*,
- zadania wykonywane przez pracownika powinny znajdować się na jego torze.

Model wyjściowy musi być modelem UML. W modelu wyjściowym tworzony jest pakiet zawierający wiele modeli *Przypadków użycia*, które odpowiadają organizacjom na diagramie procesów BPMN. Wewnątrz każdego modelu tworzone są dwa pakiety. Pierwszy z nich zawiera aktorów i ich powiązania z przypadkami użycia, a drugi przypadki użycia. Aktorzy systemowi w UML są tworzeni na podstawie torów

w BPMN. Przypadki użycia odpowiadają zadaniom, jakie pracownicy wykonują. Powiązania tworzone między aktorami a przypadkami użycia odzwierciedlają pracowników realizujących zadania. Rysunek 4 przedstawia mapowanie zdefiniowane dla transformacji BPMN2UC na najwyższym poziomie abstrakcji.



Rys. 4. Mapowanie dla transformacji BPMN2UC

Na rysunku występują dwa mapowania. Górne mapowanie odpowiada za nadanie nazwy pakietowi. Dolne mapowanie wykonuje praktycznie całą transformację i jest odpowiedzialne za utworzenie zawartości pakietu w modelu docelowym. Do dolnego mapowania podłączona jest klasa języka Java *BPMN2UC* i wywołanie operacji *mainBPMN2UC* z dwoma parametrami wejściowymi reprezentującymi model źródłowy (*Definitions*) i model docelowy (*Package*) (rys. 5).

Algorytm realizacji transformacji przebiega następująco:

— pobierane są wszystkie procesy zawarte w modelu wejściowym (obiekt *Definitions*). Dla diagramu typu proces występować będzie jeden proces;
— dla każdego procesu pobierane są tory, które reprezentują pracowników. Pod uwagę brane są jedynie te tory, które powiązane są z obiektem typu *Resource* (reprezentującym organizację);
— jeśli nie istnieje model w pakiecie wyjściowym odpowiadający elementowi typu *Resource*, to tworzony jest nowy model o takiej samej nazwie jak element *Resource*. Następnie do modelu dodawane są dwa pakiety: *Actors* i *Use Cases*. W pakiecie *Actors* dodawany jest aktor o takiej samej nazwie jak nazwa toru;
— z każdego toru pobierane są zadania (ang. *Task*). W pakiecie *Use Cases* modelu, który odpowiada elementowi *Resource*, dodawany jest przypadek użycia o takiej samej nazwie jak nazwa zadania. Następnie tworzone jest

powiązanie aktora, który powstał na bazie toru, z przypadkiem użycia, który odpowiada zadaniu z tego toru za pomocą związku asocjacji.

```
public class BPMN2UC {
      public static void mainBPMN2UC(Definitions bpmn,
Package ucPerspective){

            EList<RootElement> rootElements = bpmn.
getRootElements();
            Iterator<RootElement> rootElementsIterator =
rootElements.iterator();
            while(rootElementsIterator.hasNext()){
                  RootElement rootElement =
rootElementsIterator.next();
                  if(rootElement instanceof Process){
--- transformation body ---
}
}
private static Actor addActor(Lane lane, Package
actorsPackage
 if(actorsPackage.getPackagedElement(lane.getName()) !=
null){
      if(actorsPackage.getPackagedElement(lane.getName())
instanceof Actor){
       return (Actor) actorsPackage.
getPackagedElement(lane.getName());}
      else{
       return (Actor) actorsPackage.
createPackagedElement(lane.getName(),
 UMLPackage.eINSTANCE.getActor());}}
 else{ return (Actor) actorsPackage.
createPackagedElement(lane.getName(), UMLPackage.eINSTANCE.
getActor());}
 }
}
```

Rys. 5. Fragment klasy Java BPMN2UC odpowiadającej za transformację modelu BPMN do UML

Systemy informatyczne budowane są, aby wspierać realizację zadań w procesach biznesowych. Zatem jest wysoce prawdopodobne, że znaczna część funkcjonalności integrowanych systemów informatycznych wynika z zadań w procesach biznesowych. Dzięki takiej transformacji *Główny Analityk* uzyska komplet kandydujących przypadków użycia wspierających procesy biznesowe realizowane w organizacji. Uzyskuje się automatyczne przejście z poziomu opisu organizacji do poziomu specyfikacji wymagań na systemy informatyczne. Dalsze transformacje dotyczą przejścia z poziomu specyfikacji wymagań na poziom projektu systemu informatycznego.

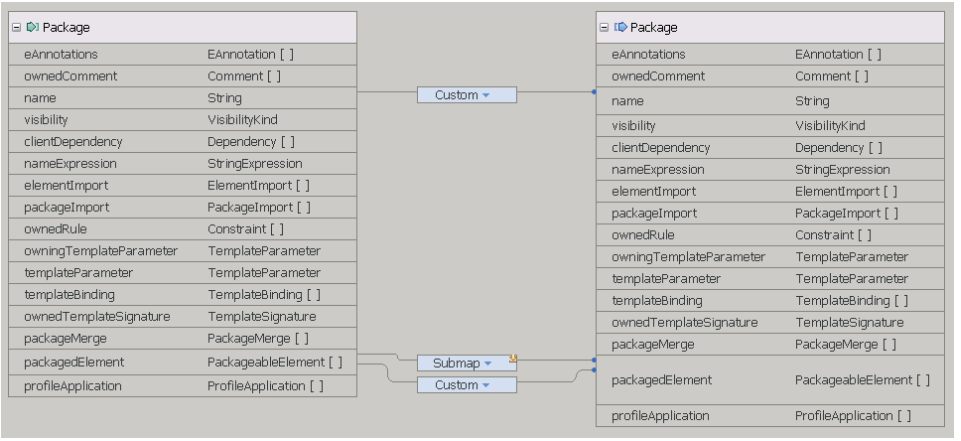
8. Transformacja widoku *Przypadków użycia* do widoku *Logiki*

Po weryfikacji i wyspecyfikowaniu przez analityków przypadków użycia możliwa staje się generacja elementów modelu *Projektowego*. Model źródłowy w tej transformacji musi być zgodny w modelem docelowym transformacji BPMN2UC. Podstawową konstrukcją w modelu *Projektowym* jest realizacja przypadku użycia. Modelowana jest ona jako kolaboracja ze stereotypem <<UseCaseRealization>>. Realizacja przypadku użycia grupuje elementy projektowe realizujące przepływ zdarzeń określony w przypadku użycia. Struktura elementów projektowych, występujących w realizacji przypadku użycia, oraz ich interakcje przedstawiane są na diagramach sekwencji, komunikacji oraz klas. Transformacja UC2Logical tworzy diagramy sekwencji i klas. Diagramy komunikacji zostały pominięte, gdyż można je automatycznie wygenerować w RSA na podstawie diagramów sekwencji. Jako wynik transformacji tworzony jest pakiet o nazwie *Design model*. Rysunek 6 przedstawia mapowanie zdefiniowane dla transformacji UC2Logical na najwyższym poziomie abstrakcji.

Pierwsze mapowanie pomiędzy elementami *name* nadaje nazwę pakietowi docelowemu. Kolejne mapowanie jest typu *Submap* pomiędzy atrybutami *packagedElement*. Atrybut *packagedElement* jest listą przechowującą wszystkie elementy wchodzące w skład pakietu. Celem tego mapowania jest odwzorowanie elementów klasy *Model* pakietu źródłowego na elementy klasy *Model* pakietu docelowego. Zrealizowano to za pomocą mapowania zagnieżdżonego. Ostatnie mapowanie typu *Custom* wypełnia pakiety docelowe odpowiednimi elementami. Do tego mapowania podłączona została klasa języka Java *UC2Logical i* wywołanie operacji *addDiagrams* z dwoma parametrami wejściowymi reprezentującymi model źródłowy (*Package*) i model docelowy (*Package*).

9. Transformacja widoku *Przypadków użycia* do widoku *Integrowanych usług*

Po weryfikacji i wyspecyfikowaniu przez analityków przypadków użycia możliwa staje się generacja konstrukcji modelowych w modelu *Usług*. Każdy model *Przypadków użycia* tworzony jest dla odrębnego systemu informatycznego. Na diagramach przypadków użycia pokazywane są przypadki użycia i aktorzy tego systemu informatycznego oraz systemy informatyczne integrowane przez magistralę usług. System informatyczny występujący w roli systemu integrowanego przez magistralę usług oznaczany jest stereotypem <<IntegratedSystem>> z profilu *UML Profile for Integration Platform* [17]. Gdy na diagramie przypadków użycia znajduje się system ze stereotypem <<IntegratedSystem>>, powiązany z przynajmniej jednym przypadkiem użycia, wygenerowane zostaną w modelu *Usług* komponenty odpowiadające



Rys. 6. Mapowanie dla transformacji UC2Logical

systemowi opisywanemu oraz integrowanemu. Ponadto utworzony zostanie komponent odpowiadający magistrali usług [9, 12] ze stereotypem <<ESB>> z profilu *UML Profile for Integration Platform*. Utworzony zostanie interfejs z operacją odpowiadającą przypadkowi użycia, z którym system integrowany pozostaje w związku asocjacji skierowanej. Stworzone zostanie także powiązanie między komponentami odpowiadającymi systemom z modelu *Przypadków użycia*. Powiązanie takie realizowane jest przez magistralę usług. W zależności od zwrotu związku asocjacji systemu integrowanego i przypadku użycia ustalony zostaje dostawca i klient usługi. Jeśli asocjacja jest skierowana do przypadku użycia, to komponent odpowiadający systemowi, w którym zawarty jest przypadek użycia, jest dostawcą usługi. Jeśli asocjacja jest skierowana do systemu integrowanego to komponent mu odpowiadający jest dostawcą usługi. Wszystkie komponenty wystawiające lub korzystające z usług opatrzone są stereotypem <<Capability>>. Komponent będący dostawcą usługi otrzymuje stereotyp <<Provider>>, a komponentowi korzystającemu z usługi nadany jest stereotyp <<Consumer>>. Są to stereotypy języka modelowania SoaML [33].

Wejściem transformacji jest model *Przypadków użycia*. Wymagane jest, aby dla każdego z modeli stworzone były pakiety *Actors* i *UseCases*. Wszystkie obiekty klasy *Actor* powinny znajdować się w pakiecie *Actors*. Przypadki użycia powinny znajdować się w pakiecie *UseCases*. Jeśli aktor w jednym z modeli reprezentuje integrowany system, to wymagane jest, aby posiadał on stereotyp <<IntegratedSystem>>, a także aby jego nazwa była identyczna z nazwą modelu reprezentującego integrowany system.

Na wyjściu transformacji tworzony jest pakiet o nazwie *Services Model*. Wewnątrz pakietu tworzony jest pakiet *Integrated services perspective*. W pakiecie *Integrated services perspective* tworzony jest komponent o nazwie *ESB* ze stereotypem <<ESB>>. Dla każdego z integrowanych systemów tworzony jest jeden

komponent o nazwie zgodnej z nazwą tego systemu. Dla każdego z przypadków użycia, który jest wykorzystywany przez zewnętrzny system, tworzony jest interfejs. Dla każdej asocjacji pomiędzy przypadkiem użycia a aktorem ze stereotypem <<IntegratedSystem>>:

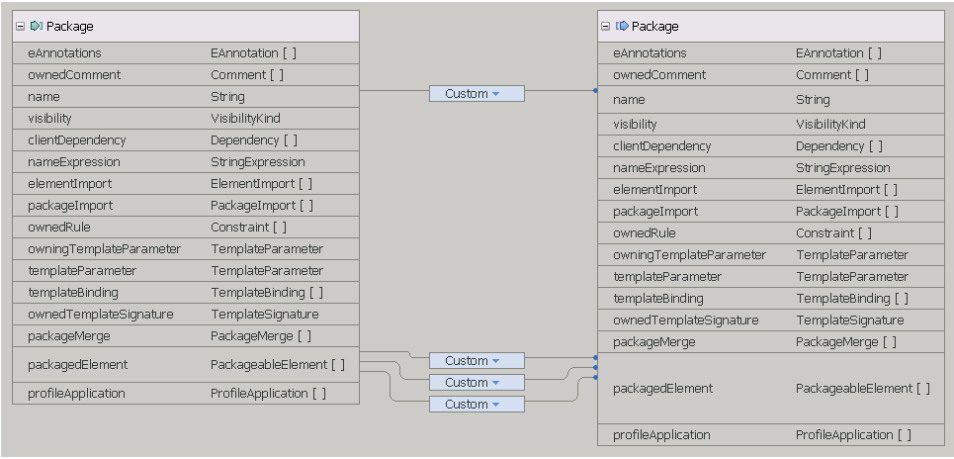
— jeśli związek skierowany jest od aktora do przypadku użycia, to:
  - tworzony jest związek *Interface Realization* od komponentu reprezentującego system realizujący przypadek użycia do interfejsu, który odpowiada przypadkowi użycia,
  - tworzony jest związek *Usage* od komponentu reprezentowanego przez aktora do interfejsu, który odpowiada przypadkowi użycia,
  - komponentowi odpowiadającemu aktorowi nadaje się stereotypy <<Capability>> i <<Consumer>>,
  - komponentowi odpowiadającemu systemowi, który realizuje przypadek użycia, nadaje się stereotypy <<Capability>> i <<Provider>>,
  - komponentowi *ESB* nadaje się związek zarówno *Usage* jak i *Interface Realization* względem interfejsu, który odpowiada przypadkowi użycia,
— jeśli związek skierowany jest od przypadku użycia do aktora, to tworzone są elementy analogiczne do tych przedstawionych powyżej, z tą różnicą, że zamieniają się role komponentu reprezentującego system realizujący przypadek użycia i komponentu, który odpowiada aktorowi,
— jeśli związek jest dwustronny, to realizowane są obydwa przedstawione powyżej warianty.

Związki typu *Interfejs Realization* wskazują interfejsy wystawiane przez komponent. Związki typu *Usage* wskazują interfejsy wymagane przez komponent. Stereotyp <<Provider>> wykorzystuje się do oznaczenia dostawcy usługi. Natomiast stereotyp <<Consumer>> stosuje się do oznaczenia usługobiorcy.

Rysunek 7 przedstawia mapowanie zdefiniowane dla transformacji UC2IS na najwyższym poziomie abstrakcji. W transformacji zastosowano cztery mapowania typu *Custom*. Pierwsze mapowanie nadaje nazwę modelowi *Services Model*. Celem drugiego mapowania jest utworzenie pakietu *Integrated services perspective* i komponentu *ESB*. W pierwszym mapowaniu wyszukiwane są również w pakiecie źródłowym profile zawierające potrzebne stereotypy: <<ESB>>, <<Capability>>, <<Provider>>, <<Consumer>>. W pakiecie *Integrated services perspective* tworzony jest komponent o nazwie *ESB* i stereotypie <<ESB>>.

Trzecia transformacja uzupełnia pakiet *Integrated services perspective* o odpowiednie komponenty. Jej działanie polega na przeszukaniu wszystkich elementów w pakiecie źródłowym. W przypadku znalezienia instancji klasy *Model* tworzony jest w pakiecie wyjściowym komponent o tej samej nazwie co instancja modelu. Algorytm ostatniej transformacji jest następujący:

— pobranie wszystkich potrzebnych stereotypów,



Rys. 7. Mapowanie dla transformacji UC2IS

— pobranie wszystkich modeli z pakietu wejściowego. Dla każdego z modeli transformacja realizuje następujące kroki:
  - pobranie wszystkich elementów z pakietów *Actors* i *UseCases*,
  - odczytanie z pobranych elementów wszystkich asocjacji, a następnie dla każdej z asocjacji wiążącej aktora ze stereotypem <<IntegratedSystem>> i posiadającej przynajmniej jeden koniec typu *Navigable* tworzony jest interfejs o takiej samej nazwie jak nazwa przypadku użycia. Tworzona jest asocjacja pomiędzy interfejsem a komponentami odpowiadającymi aktorowi i modelowi zawierającemu przypadek użycia. Zwrot i stereotyp asocjacji został opisany przy okazji omówienia wyjścia transformacji.

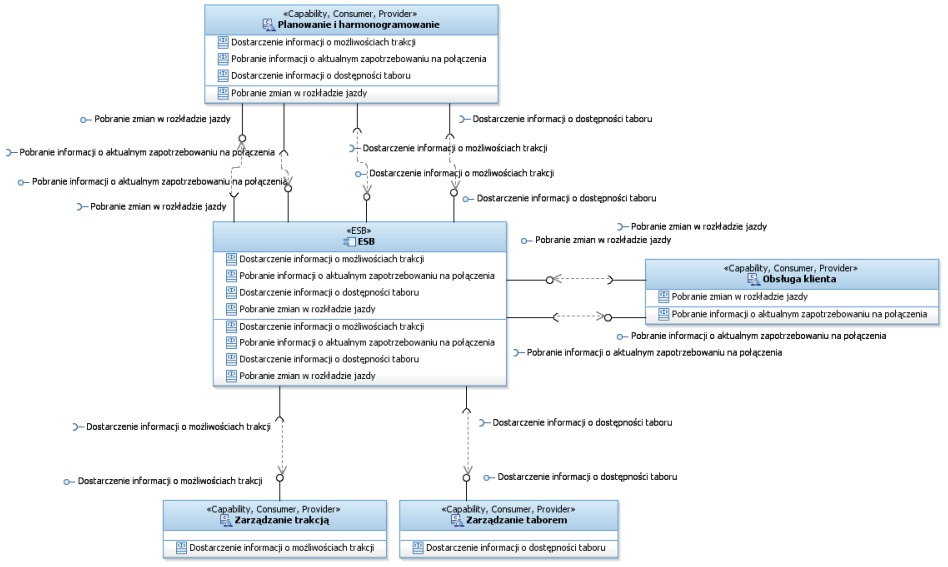
Mapowania zapisywane są w postaci pliku XML. Rysunek 8 przedstawia fragment pliku mapowania *UC2IntSrv.mapping* dotyczący mapowania tworzenia komponentu ESB oraz mapowania tworzenia interfejsów i asocjacji.

W wyniku transformacji UC2IS uzyskiwany jest zestaw usług, które magistrala usług udostępnia poszczególnym systemom informatycznym wchodzącym w skład platformy integracyjnej. Rysunek 9 przedstawia przykład diagramu komponentów w modelu *Usług* uzyskany z transformacji UC2IS.

```
- <mapping name="4">
 <input path="packagedElement" var="packagedElement_
src"/>
 <output path="packagedElement" var="packagedElement_
tgt"/>
 - <custom>
 - <code language="java">
```

```
UC_to_intserv.addEsbComponent(Package_src,Package_tgt);
 </code>
 </custom>
 </mapping>
- <mapping name="3">
 <input path="packagedElement" var="packagedElement_
src"/>
 <output path="packagedElement" var="packagedElement_
tgt"/>
 - <custom>
 - <code language="java">
UC_to_intserv.addInterfaces(Package_src,Package_tgt);
 </code>
 </custom>
 </mapping>
```

Rys. 8. Fragment pliku mapowania UC2IntSrv.mapping



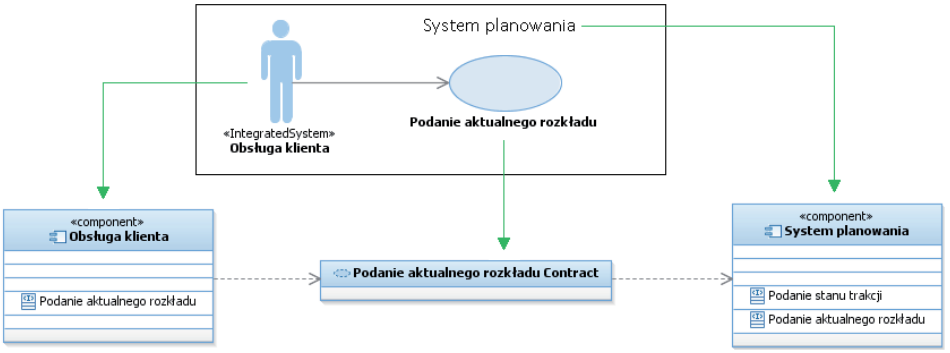
Rys. 9. Przykład diagramu komponentów uzyskanego z transformacji UC2IS

10. Transformacja widoku *Przypadków użycia* do widoku *Kontraktów*

Na podstawie zweryfikowanych i wyspecyfikowanych przypadków użycia w modelu *Przypadków użycia* możliwe staje się wygenerowanie kontraktów w modelu *Usług*. Komponenty reprezentujące systemy uzyskuje się dzięki transformacji

widoku *Przypadków użycia* do widoku *Integrowanych usług*. Dla każdego związku pomiędzy przypadkiem użycia a aktorem ze stereotypem <<IntegratedSystem>> tworzony jest kontrakt i nadawana jest mu nazwa przypadku użycia. Kontrakt ten wiązany jest związkiem *Usage* z komponentami, które odpowiadają systemowi integrowanemu i systemowi realizującemu przypadek użycia. Rysunek 10 przedstawia przykład transformacji widoku *Przypadków użycia* do widoku *Kontraktów*.

W transformacji tej wykonywane jest jeszcze jedno istotne zadanie. Niezależnie budowane i rozwijane systemy informatyczne najczęściej posiadają różne struktury danych, formaty komunikatów i protokoły komunikacyjne. W celu umożliwienia współpracy między systemami informatycznymi stosuje się przepływy mediacyjne z zastosowaniem wzorców mediacyjnych [16]. W tym celu w modelu *Usług* stosuje się diagramy przepływów mediacyjnych, które są rozszerzeniem diagramów aktywności języka UML i stanowią autorską propozycję autora artykułu [15, 16, 18]. W modelu *Usług* dla każdego kontraktu tworzony jest diagram przepływów mediacyjnych.



Rys. 10. Przykład transformacji elementów z widoku *Przypadków użycia* do widoku *Kontraktów*

11. Automatyzacja tworzenia struktur modeli

W wyniku opisanych transformacji uzyskujemy zestaw diagramów i elementów modelowych. Istotne jest pogrupowanie tych elementów w odpowiednie modele i pakiety. Zaproponowane transformacje tworzą tego typu struktury do grupowania elementów w celu zarządzania złożonością modeli oraz zapewnienia spójności zbioru elementów modelowych. Dla przykładu, w wyniku transformacji *UC2Contracts* oraz transformacji *UC2IS* tworzony jest model prezentujący dwa widoki architektoniczne: *Integrowanych usług* oraz *Kontraktów*. Elementy istotne z punktu widzenia każdego widoku są grupowane w dwóch pakietach, których nazwy pochodzą od widoku, który reprezentują. W pakiecie *Integrowane usługi* umieszczane są wszystkie utworzone

komponenty oraz interfejsy. Ważne jest, że w pakiecie zawarte są także informacje o związkach zachodzących pomiędzy komponentami i interfejsami. Związki, w jakich występują komponenty, podłączane są bezpośrednio do komponentów. W wygenerowanej strukturze należy uszczegółowić interfejsy, które odpowiadają przypadkom użycia. Jest wysoce prawdopodobne, że operacje realizowane między komponentami będą miały mniejszą granulację niż poziom przypadków użycia.

12. Podsumowanie i kierunki dalszych prac

W artykule przedstawiono zaprojektowane i zaimplementowane transformacje automatyzujące projektowanie architektury platformy integracyjnej. Transformacje obejmują poziomy: procesów biznesowych, specyfikacji wymagań oraz projektu systemu. Dzięki transformacjom uzyskiwane są modele, diagramy, elementy modelowe oraz powiązania pomiędzy nimi. Ponadto transformacje grupują elementy modelowe w odpowiednie pakiety. Transformacje zostały zaprojektowane tak, aby umożliwiały generację elementów modelowych zgodnie z modelem widoków architektonicznych „1 + 5" dostosowanym do opisu rozwiązań integracyjnych. Wykorzystano autorskie profile języka UML: *UML Profile for Integration Platform* [17] oraz *UML Profile for Integration Flows* [16].

Dzięki zastosowaniu opracowanych transformacji możliwe jest znaczne skrócenie czasu tworzenia opisu architektonicznego i uniknięcie dużej liczby błędów. Szczególnie ma to znaczenie przy projektowaniu złożonych rozwiązań integracyjnych wielu systemów informatycznych. Analizy dostępne dla systemów informatycznych projektowanych w sektorze medycznym pokazują, że w projektach, gdzie stosowana jest inżynieria sterowana modelami, notuje się trzykrotne skrócenie czasu wytwarzania oprogramowania [19, 35]. Przy projektowaniu rozwiązań informatycznych dla zastosowań w sieci Internet oszczędności z zastosowania inżynierii sterowanej modelami są ponad dwukrotne (skrócenie czasu wytwarzania oprogramowania o 59%) [25]. Ponadto, dzięki zastosowaniu transformacji, zapewnione jest utrzymanie kompletności opisu architektonicznego oraz spójności elementów między modelami. Badania pokazują, że dzięki zastosowaniu transformacji notuje się 10-krotny spadek w liczbie zgłaszanych błędów w oprogramowaniu [19, 35]. Możliwość skrócenia czasu wytwarzania oprogramowania i podniesienia jego jakości powodują, że modele stają się podstawowym artefaktem w procesie wytwarzania oprogramowania. Dlatego bardzo istotnym i aktualnym zagadnieniem staje się zapewnienie odpowiedniego poziomu jakości opracowywanych modeli [2, 22]. Do tej pory główny nacisk kładziony był w badaniach na zapewnienie jakości transformacji przez ich weryfikację i walidację [3, 6, 7, 20]. W literaturze przedmiotu pojawiają się także prace dotyczące zastosowania inżynierii sterowanej modelami do automatycznego generowania transformacji modeli [4].

Ciekawym kierunkiem dalszych prac byłoby opracowanie metody zapewnienia jakości transformowanym modelom oraz odpowiedniej jakości samej transformacji. W ramach dalszych prac planowane jest uelastycznienie transformacji *BPMN2UC*, tak aby wykorzystywała ona jako element źródłowy plik XML. Dzięki takiej zmianie transformacja *BPMN2UC* mogłaby transformować modele procesów tworzone w dowolnym narzędziu zapisującym model BPMN w postaci XML. Kolejnym istotnym rozszerzeniem mogłoby być dodanie do zaprojektowanych transformacji relacji śledzenia i umożliwienie aktualizacji modelu docelowego w przypadku modyfikacji modelu źródłowego [32]. Dalszym rozszerzeniem byłaby modyfikacja zaproponowanych transformacji tak, aby stały się dwukierunkowe [28]. Ponadto planowane jest dodanie transformacji typu model w kod i generacja kodu źródłowego klas, usług i przepływów mediacyjnych z opracowanych modeli.

Artykuł został opracowany na podstawie referatu wygłoszonego na Konferencji Naukowej pt. „System Engineering 2013".

LITERATURA

[1] M. Abi-Antoun, J. Aldrich, N. Nahas, B. Schmerl, D. Garlan, *Differencing and merging of architectural views*, Automated Software Engineering, 15, 2008, 35-74.

[2] T. Arendt, G. Taentzer, *A tool environment for quality assurance based on the Eclipse Modeling Framework*, Automated Software Engineering, 20, 2013, 141-184.

[3] J. Bézivin, F. Jouault, *Using ATL for checking models*, Electronic Notes In Theoretical Computer Science, 152, 2006, 69-81.

[4] V.A. Bollati, J.M. Vara, Á. Jiménez, E. Marcos, *Applying MDE to the (semi-)automatic development of model transformations*, Information and Software Technology, 55, 2013, 699-718.

[5] Business Process Model and Notation (BPMN) 2.0, OMG 2011, http://www.omg.org/ spec/ BPMN/2.0/ (23.04.2013).

[6] J. Cabot, R. Clarisó, E. Guerra, J. de Lara, *Verification and validation of declarative model-to-model transformations through invariants*, The Journal of Systems and Software, 83, 2010, 283-302.

[7] D. Calegari, N. Szasz, *Verification of model transformations a survey of the State-of-the-art*, Electronic Notes In Theoretical Computer Science, 292, 2013, 5-25.

[8] M. Chinosi, A. Trombetta, *BPMN: An introduction to the standard*, Computer Standards & Interfaces, 34, 2012, 124-134.

[9] D. Chappell, *Enterprise Service Bus*, O'Reilly, 2004.

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

[11] Ecore Package API Javadoc — http://download.eclipse.org/modeling/emf/emf/javadoc /2.6.0/ org/eclipse/emf/ecore/package-summary.html (23.04.2013).

[12] T. Erl, *Service-Oriented Architecture: Concepts, Technology and Design*, Prentice Hall, 2005.

[13] A. Frece, M.B. Juric, *Modeling functional requirements for configurable content- and context-aware dynamic service selection in business process models*, Journal of Visual Languages and Computing, 23, 2012, 223-247.

[14] T. Górski, *Architectural view model for an integration platform*, Journal of Theoretical and Applied Computer Science, 6, 1, 2012, 25-34.

[15] T. Górski, *Architektura platformy integracyjnej dla elektronicznego obiegu recept*, Roczniki Kolegium Analiz Ekonomicznych, z. 25, Warszawa, 2012.

[16] T. Górski, *Platformy integracyjne. Zagadnienia wybrane*, PWN, Warszawa, 2012.

[17] T. Górski, *Profil „UML Profile for Integration Platform" do modelowania architektury platformy integracyjnej*, Inżynieria Oprogramowania w Procesach Integracji Systemów Informatycznych, PWNT, Gdańsk, 2011.

[18] T. Górski, *Projektowanie platform integracyjnych w architekturze zorientowanej na usługi*, Wiadomości Górnicze, 7-8, 2012.

[19] J.F. Groote, A.A.H. Osaiweran, J.H. Wesselius, *Analyzing the effects of formal methods on the development of industrial control software*, IEEE ICSM 2011, USA, 2011, 467-472.

[20] E. Guerra, J. de Lara, M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schönböck, W. Schwinger, *Automated verification of model transformations based on visual contracts*, Automated Software Engineering, 20, 2013, 5-46.

[21] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, *ATL: A model transformation tool*, Science of Computer Programming, 72, 2008, 31-39.

[22] M. Kessentini, H. Sahraoui, M. Boukadoum, *Example-based model-transformation testing*, Automated Software Engineering, 18, 2011, 199-224.

[23] B. Kitchenham, *Procedures for performing systematic reviews*, Keele University Technical Report TR/SE-0401, UK, 2004.

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

[25] N. Koch, A. Knapp, S. Kozuruba, *Assessment of Effort Reduction due to Model-to-Model Transformations in the Web Domain*, Lecture Notes in Computer Science, 7387, 2012, 215-222.

[26] H.S. Lahman, *Model-Based Development: Applications*, Pearson Education Inc., 2011.

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

[28] S. Mazanek, M. Hanus, *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] T. Mens, P. van Gorp, *A taxonomy of model transformation*, Electronic Notes In Theoretical Computer Science, 152, 2006, 125-142.

[30] A. Rensink, R. Nederpel, *Graph Transformation Semantics for a QVT Language*, Electronic Notes in Theoretical Computer Science, 211, 2008, 51-62.

[31] N. Rozanski, E. Woods, *Software Systems Architecture. Working with stakeholders using Viewpoints and Perspectives*, Addison Wesley, 2005.

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

[33] *Service oriented architecture Modeling Language (SoaML) Specification for the UML Profile and Metamodel for Services (UPMS) v. 1.0*, 2008.

[34] Unified Modeling Language Specification Version 2.4.1, OMG 2011, (http://www.omg.org/spec/UML/2.4.1/) (23.04.2013).

[35] M.G.J. Van den Brand, J.F. Groote, *Advances in Model Driven Software Engineering*, ERCIM News, 91, 2012.

T. GÓRSKI

Transformations for automation of integration platform's architecture designing

Abstract. The paper presents transformations that automate integration platform's architecture design. In the paper, integration platform's architecture is described in accordance with the architectural views model "1+5". Transformations of a model-to-model type were used to automate designing of integration platform architecture. The paper contains an introduction to transformations and overview of the current literature. In the paper, there are presented transformations between models in the following architectural views: Integrated Processes, Use Cases, Logical and Integrated Services. Design and implementation of transformations were performed in an IBM Rational Software Architect. In conclusion, the paper presents the advantages of architecture designing automation for information systems development projects and further works.

Keywords: information systems integration, transformations, model-driven engineering