Związki pomiędzy modelami w metodyce USDP

Bogumiła Hnatkowska

Zbigniew Huzar

Lech Tuzinkiewicz
Wydziałowy Zakład Informatyki
Politechniki Wrocławskiej

Streszczenie

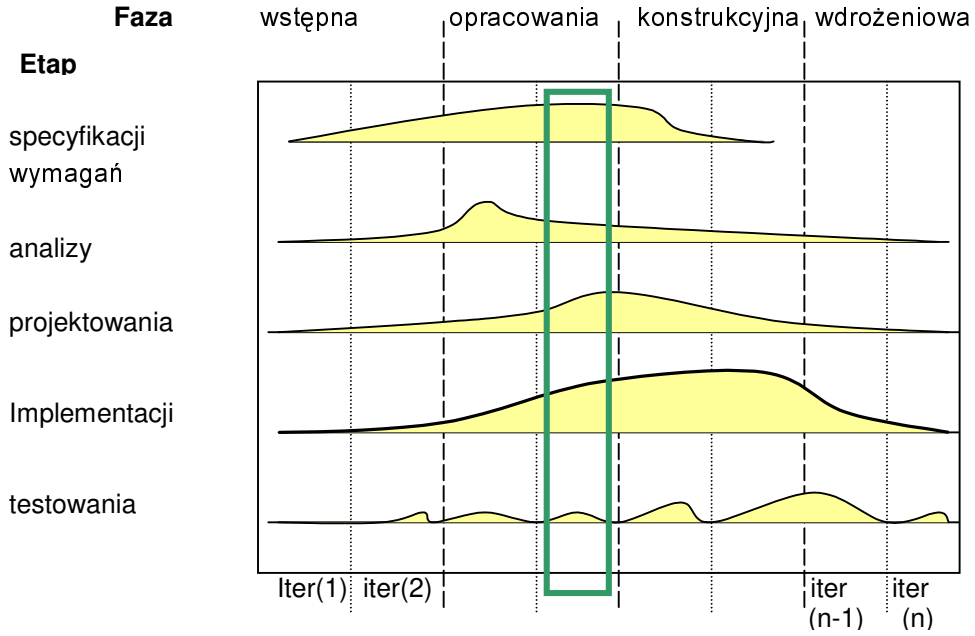
Artykuł charakteryzuje modele powstające w procesie wytwarzania oprogramowania z zastosowaniem metodyki USDP. Modele te reprezentują system na różnych poziomach abstrakcji i (lub) w różnych okresach jego życia. Praca omawia związki zgodności pomiędzy modelami oraz określa je przez identyfikację związków zachodzących pomiędzy elementami modeli.

1. Wstęp

USDP (*Unified Software Development Process*) jest metodyką, która w kompleksowy sposób próbuje ująć główne aspekty związane z wytwarzaniem oprogramowania [4]. Kompleksowość odnosi się – jak określa się to hasłowo – do *czterech P*: *personelu* (zamawiających, wykonawców i przyszłych użytkowników), *projektu* (organizacji i zarządzania zasobami niezbędnymi do wytworzenia produktu), *produktu* (wynikowych artefaktów) oraz do *procesu* (czynności i sposobów postępowania, polegających na przekształceniu wymagań użytkownika w końcowy produkt). Metodyka przyjmuje, że proces wytwarzania składa się z czterech faz (*phases*). W każdej fazie proces można rozpatrywać w kategorii etapów (*workflows*). Wyróżnia się pięć etapów, które mogą być iteracyjnie powtarzane (rys. 1).

Proces wytwarzania jest przedstawiany jako proces modelowania – tworzenia ciągu modeli. Model jest semantycznie spójną abstrakcją systemu, czyli pewnym uproszczonym obrazem wycinka rzeczywistości. Uproszczenie jest dokony-

wane z pewnej perspektywy (punktu widzenia lub widoku) i polega na ustaleniu interesujących aspektów modelowanej rzeczywistości oraz szczegółowości ich przedstawienia. Metodyka ustala dobór perspektyw i kolejność ich stosowania. USDP bierze pod uwagę pięć perspektyw: przypadków użycia (*use-case view*), analizy (*logical view*), projektowania (*process view*), architektury (*deployment view*) oraz implementacji (*implementation view*).



Rysunek 1: Schemat wykonywanych czynności w ramach USDP

Każda perspektywa wyznacza pewien model, który jest przedstawiany w języku UML [6], [7]. Modelowanie z wykorzystaniem języka UML wymaga wsparcia narzędziowego. Narzędzia typu CASE umożliwiają edycję i zapamiętywanie modeli, ale powinny także umożliwiać sprawdzanie własności modeli. Od każdego modelu oczekuje się zgodności wewnątrz-modelowej, a od kolejnych modeli wymaga się zgodności między-modelowej. Wymogi te są związane z analizą modeli w aspekcie funkcjonalnym i są konieczne dla zapewnienia poprawności procesu wytwarzania oprogramowania. Obecnie istniejące narzędzia CASE takich możliwości nie posiadają. Przykładową analizę odnoszącą się do zgodności wewnątrz-modelowej przedstawiono w [3]. Rozważania w niniejszym artykule odnoszą się do problemów zgodności między-modelowej.

Metodyka USDP określa ogólnie modele odpowiadające poszczególnym perspektywom. Model jest określany przez zbiór elementów modelowania, które mogą w nim wystąpić. W artykule, w celu zwartej prezentacji głównych idei, dokonuje się pewnych uproszczeń, a także arbitralnych uściśleń tych modeli. Przyj-

muje się, że model składa się z elementów podstawowych, uznawanych za niezbędne, oraz elementów pomocniczych, które nie zawsze muszą wystąpić. Elementami, które bierze się pod uwagę są diagramy lub pojedyncze elementy modelowania języka UML.

Pomiędzy modelami USDP występuje relacja abstrakcji. Pojęcie relacji abstrakcji zostało wprowadzone w wersji UML 1.4 [6]. Relacja ta jest szczególnym rodzajem (specjalizacją) relacji zależności, występującej już w poprzedniej wersji 1.3 języka [7]. Jest to relacja, która – z definicji – zachodzi pomiędzy dwoma elementami (grupami elementów), opisującymi ten sam wycinek rzeczywistości na różnych poziomach abstrakcji lub z różnych punktów widzenia. Zachodzenie relacji pomiędzy modelami oznacza, że zachodzi zgodność opisu dowolnego aspektu modelowanej rzeczywistości, który jest uwzględniony przez oba modele. Pojęcie zgodności musi być precyzowane zależnie od elementów, które są użyte do opisu danego aspektu.

W pracy pominięto opis modelu testowania USDP. Artefakty tego modelu (procedury testowe, czy komponenty testowe), opisują zupełnie inny widok systemu w stosunku do pozostałych modeli.

2. Modele

2.1. Model przypadków użycia

Perspektywa przypadków użycia jest związana z etapem specyfikacji wymagań i ma na celu ustalenie z przyszłym użytkownikiem funkcji systemu. Model przypadków użycia jest przeznaczony do identyfikacji wymagań funkcjonalnych i niefunkcjonalnych tworzonego systemu – ma określać usługi oraz ich jakość. Ma jednocześnie umożliwiać jednoznaczne rozumienie dziedziny, w której system funkcjonuje. Dzięki temu, stanowiąc formę porozumienia pomiędzy wykonawcą i zleceniodawcą, umożliwia ekspertom dziedziny i zleceniodawcy weryfikację ustaleń wyspecyfikowanych przez wykonawcę. Porozumienie pomiędzy wykonawcą i zamawiającym daje podstawę do opracowania planu zarządzania przedsięwzięciem informatycznym, którego częścią główną jest wytworzenie systemu oprogramowania.

Model przypadków użycia jest artefaktem niezbędnym do realizacji następnych etapów prac, a w szczególności umożliwia zaplanowanie testów, które mają być realizowane jako etap końcowy danej iteracji.

Podstawowymi elementami modelu są: diagramy przypadków użycia (zbiór przypadków użycia i aktorów z wiążącymi je relacjami oraz słownik pojęć dziedzinowych. Przypadki użycia mają tu zwykle opis tekstowy, chociaż mogą rów-

nież mieć opis bardziej szczegółowy w postaci zbioru scenariuszy współdziałania pomiędzy użytkownikiem a systemem.

Pomocniczymi elementami modelu są: pakiety przypadków użycia, diagramy sekwencji, diagramy aktywności, diagramy stanów oraz prototypy interfejsów użytkowników. Wymienione diagramy są różnymi formami precyzyjnego wyrażania zachowań związanych z przypadkami użycia. Przy tym diagramy sekwencji pozwalają na ogół tylko na częściową specyfikacją zachowań, podczas gdy diagramy aktywności i stanów pozwalają na specyfikację pełną. Pakiety przypadków użycia służą tylko do pogrupowania (strukturalizacji) przypadków użycia, co może ułatwiać organizację prac w dalszych etapach. Prototypy interfejsów, przydatne w celu uściślenia interakcji pomiędzy użytkownikiem a systemem, nie są elementem języka UML, a zależą od wybranej metodyki i wykorzystywanych narzędzi. Taką możliwość stwarza na przykład implementacja metodyki USDP systemu RUP [5].

2.2. Model analizy

Model analizy opisuje realizację przypadków użycia. Podstawą do jego konstrukcji jest model przypadków użycia oraz wiedza analityka, który ma do dyspozycji pewien repertuar środków (na ogół nie definiowanych jawnie). W czasie budowy modelu analizy mogą ujawnić się braki we wcześniejszym modelu. Wówczas konstrukcji modelu analizy towarzyszy pozyskanie dodatkowych informacji, w celu zrozumienia i określenia zachowania systemu. Realizację przypadków użycia wyraża się przez określenie obiektów i sposobu ich współdziałania.

Podstawowymi elementami modelu analizy są: pakiety analizy, klasy analizy oraz realizacje przypadków użycia. Pakiety analizy służą grupowaniu powiązanych elementów – są abstrakcją podsystemów z modelu projektowego. Pakiety analizy mogą tworzyć wielowarstwową strukturę. Klasa analizy jest elementem, który realizuje funkcjonalność przypadku użycia. W modelu analizy mogą występować klasy opatrzone jednym z trzech stereotypów: «boundary», «control», «entity». Realizacja przypadku użycia jest opisem współdziałania obiektów w ramach konkretnego przypadku użycia (a dokładniej jego instancji – scenariusza). Do opisu wykorzystuje się diagramy współdziałania, które przedstawiają powiązania obiektów oraz ich zachowanie podczas realizacji konkretnych scenariuszy.

2.3. Model projektowy

Model projektowy jest uściśleniem modelu analizy. Również opisuje realizację przypadków użycia, ale na wyższym poziomie szczegółowości. Ponadto,

uwzględnia aspekty związane z realizacją wymagań niefunkcjonalnych, w tym, na przykład, z wybranym środowiskiem implementacyjnym, wykorzystywanym oprogramowaniem (również spadkowym), itp.

Podstawowymi elementami modelu projektowego są podsystemy, interfejsy, klasy projektowe i projektowe realizacje przypadków użycia. Podsystem dziedziczy własności pakietu i klasyfikatora. Reprezentuje jednostkę zachowania, która udostępnia i realizuje zbiór operacji. Architektura systemu (wysokiego poziomu) jest opisana w terminach podsystemów i ich interfejsów. Wskazuje również zależności pomiędzy podsystemami. Interfejs jest nazwanym zbiorem operacji. Zarówno podsystem, jak i klasa projektowa mogą realizować wiele interfejsów. Jeden interfejs może być implementowany przez wiele klas i (lub) podsystemów. Podsystemy, podobnie do pakietów analizy, mogą tworzyć wielowarstwowe struktury. Klasa projektowa jest abstrakcją klasy lub podobnego elementu, występującego w języku implementacji. Powinna być wyspecyfikowana z dokładnością do sygnatur atrybutów i operacji oraz wszystkich asocjacji. W modelu projektowym podejmuje się decyzje dotyczące wprowadzenia elementów współbieżnych – klas aktywnych – oraz sposobu ich komunikacji (synchroniczna, asynchroniczna). Projektowa realizacja przypadku użycia jest wyrażona, podobnie jak w modelu analizy, przy pomocy diagramów interakcji oraz diagramów klas. W modelu projektowym kładzie się nacisk na aspekt czasowy współdziałania, stąd zaleca się w tym etapie wykorzystanie diagramów sekwencji.

Elementami pomocniczymi modelu projektowego są diagramy stanów, budowane dla klas (zwykle dla klas aktywnych), oraz diagramy aktywności, wykorzystywane, na przykład, do ilustracji przepływu sterowania i danych w operacji klasy.

2.4. Model rozmieszczenia

Model rozmieszczenia ilustruje aspekt fizyczny systemu. Pokazuje, jak elementy oprogramowania są powiązane ze środowiskiem wykonawczym.

Podstawowymi elementami modelu rozmieszczenia są węzły i komponenty. Węzeł jest abstrakcją (modelem) elementu obliczeniowego, na przykład, komputera. Komponent reprezentuje pewien zasób programowy, np. bibliotekę lub program wykonywalny. Komponenty umieszczone w różnych węzłach mogą się komunikować, wykorzystując połączenia istniejące pomiędzy węzłami, w których są osadzone.

Model rozmieszczenia różni się od pozostałych modeli tym, że jest on opracowywany w dwóch etapach: projektowania i implementacji.

Budowę modelu rozmieszczenia rozpoczyna się na etapie projektowania, w ramach projektu architektonicznego. Wówczas identyfikowane są węzły oraz ich

konfiguracje sieciowe. Wstępnie też do węzłów są alokowane klasy aktywne. Model rozmieszczenia jest uściślany podczas budowany modelu implementacyjnego. Wówczas dokonuje się alokacji komponentów na węzły.

2.5. Model implementacyjny

Model implementacyjny opisuje projektowany system w terminach podsystemów implementacyjnych, komponentów oraz interfejsów. Podsystemy implementacyjne oraz ich interfejsy są odwzorowywane wzajemnie jednoznacznie z podsystemów projektowych. Sposób odwzorowania zależy od docelowego środowiska implementacyjnego. Na przykład, gdy docelowym językiem implementacji jest Java, podsystem projektowy może być odwzorowany do pakietu, a interfejs projektowy, do interfejsu Javy. Komponent reprezentuje zasób programowy (plik) i zwykle zawiera implementację jednej lub kilku klas projektowych. Podobnie do podsystemu, może zawierać elementy, które realizują wiele interfejsów.

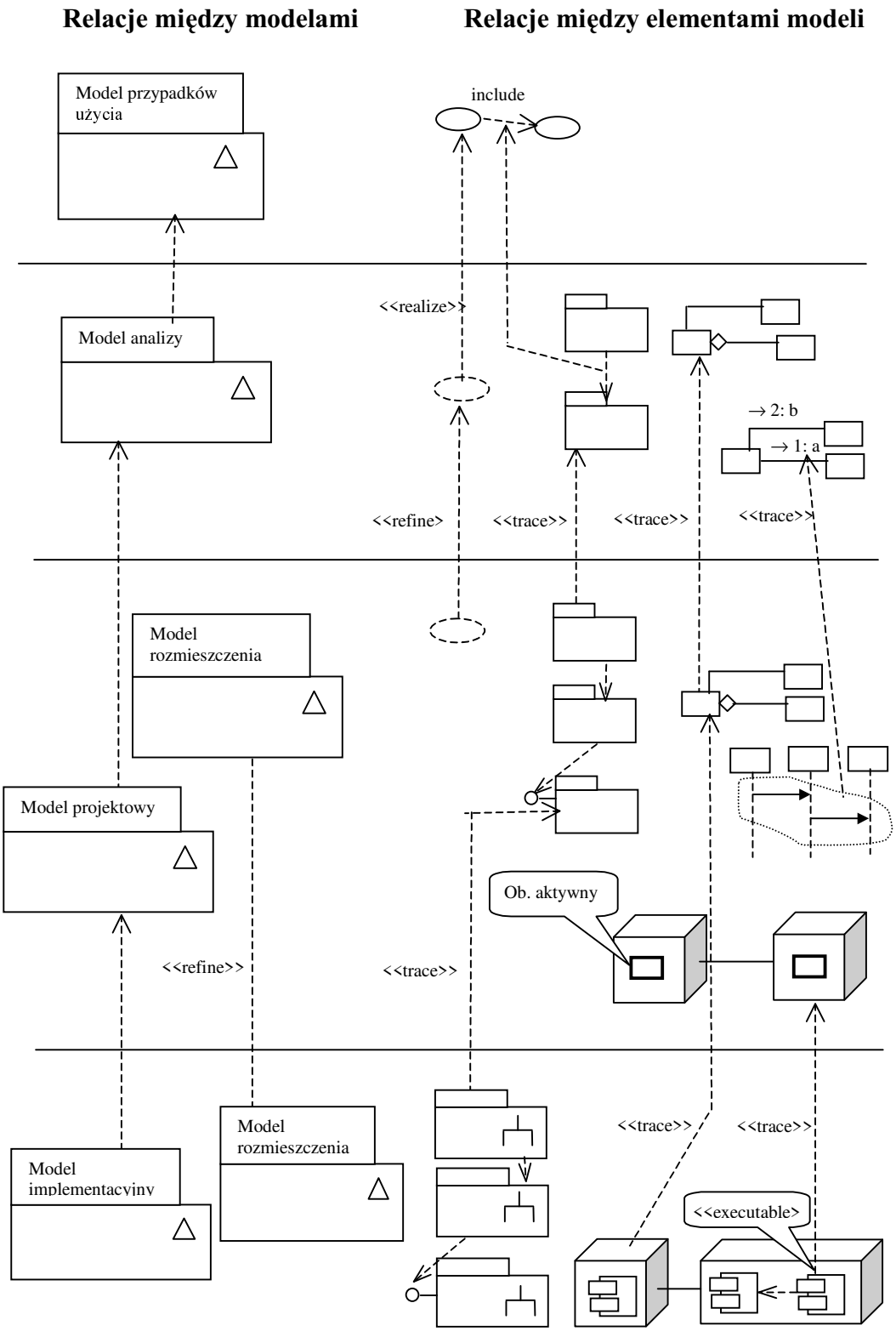
3. Związki pomiędzy modelami

Pomiędzy modelami, budowanymi w kolejnych etapach, w ramach jednej iteracji, powinna zachodzić relacja abstrakcji. Relacja ta może być opatrzona następującymi standardowymi stereotypami: «trace», «refine», «realize» oraz «derive». Stereotyp «trace» służy do wskazania historii elementu; są nim powiązane elementy modelu reprezentujące różne widoki tego samego pojęcia. Stereotyp «refine» pozwala wskazać związek pomiędzy elementem ogólnym i jego uszczegółowieniem. Stereotyp «realize» łączy element będący specyfikacją z elementem, który jest realizacją tej specyfikacji. Niektóre elementy modelu, na przykład, asocjacje, atrybuty, mogą zostać wyprowadzone z innych – tego samego typu – elementów modelu. Stereotyp «derive» pozwala wskazać zależność pomiędzy elementem wyprowadzonym i jego elementami źródłowymi. Nie jest on brany pod uwagę przy definicji związków zachodzących pomiędzy modelami.

Rysunek 2 przedstawia najistotniejsze związki zachodzące pomiędzy modelami USDP oraz pomiędzy ich podstawowymi elementami. W notacji UML 1.4 pakiety opatrzone symbolem Δ oznaczają modele, natomiast opatrzone symbolem oznaczają podsystemy.

Stereotypem «trace» są – z definicji – opatrzone wszystkie te związki pomiędzy modelami, pomiędzy którymi zachodzi dowolna relacja abstrakcji – dlatego pominięto je w ilustracji.

Model analizy jest realizacją modelu przypadków użycia. Dlatego związek pomiędzy przypadkami użycia (w modelu przypadków użycia) oraz ich realizacjami (w modelu analizy) jest etykietowany stereotypem «realize».



Rysunek 2: Związki pomiędzy modelami i ich elementami

Zależności zachodzące pomiędzy przypadkami użycia (np. «include») mogą znaleźć swoje odzwierciedlenie w zależnościach zachodzących pomiędzy pakietami analizy. Ma to miejsce, na przykład, w sytuacji, gdy dla każdego przypadku użycia zidentyfikowano jeden pakiet analizy.

Model projektowy jest uściśleniem modelu analizy, przy czym uściślenie to ma raczej charakter strukturalny. Wprowadza się nowe elementy lub uszczegóławia istniejące bez zmiany modelowanej funkcjonalności.

Każda klasa modelu analizy jest abstrakcją pewnego zbioru klas projektowych (być może pustego, gdy zdecydujemy się delegować zakres odpowiedzialności różnych klas analizy do tej samej klasy projektowej). Przykładowo, klasa ze stereotypem «boundary», powiązana z aktorem, może na etapie projektu zostać zastąpiona zbiorem klas, reprezentujących okna w środowisku graficznym.

Pakiet analizy jest abstrakcją podsystemu projektowego. Dla każdego pakietu modelu analizy jest identyfikowany co najmniej jeden podsystem projektowy. Dodatkowo, model projektowy może wprowadzać dodatkowe podsystemy, zwłaszcza z dolnych warstw architektury.

Komunikaty wymieniane przez obiekty klas analizy mogą reprezentować złożone współdziałanie na etapie projektu. Na przykład, pojedyncza komunikacja pomiędzy obiektami klas analizy może być reprezentowana ciągiem komunikacji pomiędzy obiektami klas projektowych.

Model rozmieszczenia budowany na etapie projektu nie jest powiązany relacją abstrakcji z modelem analizy. Jest on uściślany na etapie implementacji, przez wskazanie odwzorowania komponentów do węzłów.

Do podstawowych elementów modelu implementacyjnego należą komponenty. Każdy komponent jest fizyczną reprezentacją co najmniej jednej klasy modelu projektowego.

Architektura modelu implementacyjnego jest wyrażona w terminach podsystemów implementacyjnych, które zostały wyprowadzone w sposób wzajemnie jednoznaczny z podsystemów projektowych. W tym zakresie można mówić o transformacji modelu projektowego do modelu implementacyjnego.

4. Podsumowanie

W pracy przedstawiono modele powstające podczas opracowywania oprogramowania z wykorzystaniem metodyki USDP i omówiono związki zachodzące pomiędzy tymi modelami. Zwrócono uwagę na te związki, które mogą być podstawą do stwierdzania czy modele są zgodne. Oczywista obserwacja, że dwa modele powinna wiązać relacja abstrakcji jest punktem wyjścia do badania związków pomiędzy elementami modeli. Pomiędzy elementami modeli zachodzą szczególne rodzaje relacji abstrakcji, spośród których najistotniejsze są relacje określone

przez stereotypy «refine» oraz «realize». Znaczenie każdego z tych stereotypów wymaga jeszcze dalszego uściślenia w zależności od tego, do jakiego rodzaju elementów mają się odnosić. Uściślenie powinno uwzględniać pragmatykę w zakresie osiągalnego stopnia kompletności modelu na danym etapie i jednocześnie być na tyle precyzyjne, aby zachodzenie lub nie zachodzenie relacji można badać algorytmicznie.

Jako bardzo prosty przykład takiego uściślenia rozważmy związek pomiędzy pojedynczym przypadkiem użycia, z modelu przypadków użycia, a realizującą go grupą współdziałania, z modelu analizy.

Każdy przypadek użycia *PU* powinien mieć opis tekstowy. Dodatkowo, mając na uwadze potrzebę badania relacji między modelami, wymaga się uzupełnienia takiego opisu o charakterystykę stworzoną w postaci podzbioru słów kluczowych *opis*(*PU*) – podzbioru pojęć ze słownika dziedzinowego *SD*. Podobny opis *opis*(*GW*) powinna posiadać grupa współdziałania *GW* realizująca *PU*. W przypadku istnienia takich opisów koniecznym warunkiem zgodności jest $opis(PU) \subseteq opis(GW)$.

Każda klasa analizy *KA* występująca w grupie współdziałania *GW* powinna odzwierciedlać pojęcia reprezentowane w słowniku pojęć dziedzinowych. Stąd też każdej klasie należy przypisać co najmniej jeden atrybut, a zbiór nazw atrybutów *atr*(*KA*) klasy *KA* powinien być podzbiorem pojęć dziedzinowych, czyli $atr(KA) \subseteq SD$. Zakładamy tu, że nazwy klas są tylko identyfikatorami, chociaż nazwy te także mogą być elementami słownika *SD*.

Dodatkowo, nazwy asocjacji, nazwy ról i nazwy komunikatów, *nazwy*(*GW*), jakie występują w grupie współdziałania *GW* powinny również należeć do pojęć dziedzinowych, czyli $nazwy(GW) \subseteq SD$.

Wynikającym stąd warunkiem wewnętrznej zgodności w obrębie samej grupy współdziałania jest zależność:

$$\{atr(KA) \mid KA \in GW\} \cup nazwy(GW) \subseteq opis(GW).$$

Za podstawę przedstawianych rozważań została przyjęta metodyka USDP i jej implementacja w systemie RUP [5]. Z uwagi na swoje walory zasługuje ona na wyróżnienie. Niestety nie rozstrzyga ona jednak problemu zgodności modeli (ze względu na swoją generyczność), co jest w ogóle charakterystyczne dla metodyk opartych na podejściu obiektowym do tworzeniu oprogramowania, na przykład [1], [2]. Ścisłe zdefiniowanie rozpatrywanych relacji może stanowić specyfikację dodatkowych funkcji, które mogłyby realizować narzędzia CASE wspomagające wytwarzanie oprogramowania. Stąd nasuwa się wniosek o sensowności dalszych badań w kierunku identyfikacji i uściślania związków pomiędzy elementami tworzonych modeli.

Bibliografia

[1] Douglass B.P., *Doing Hard Real. Developing Real-Time Systems with UML*, Addison-Wesley, 1999

[2] D'Souza D.F., Wills A.C., *Objects, Components and Frameworks with UML. The Catalysis™ Approach*, Addison-Wesley, 1999

[3] B. Hnatkowska, Z. Huzar, J. Magott, *Consistency Checking in UML Models*, w: J. Zendulka (ed.), Proceedings of the Conference "Information Systems Modelling", 35-40, Ostrava 2001

[4] I. Jacobson, G. Booch, J. Rambough, *The Unified Software Development Process*, Addison-Wesly, 1999

[5] P. Kruchten, *The Rational Unified Process. An Introduction,* Addison-Wesly, 1999

[6] *OMG Unified Modeling Language Specification*, *Version 1.4*, Rational Software Corporation, November 2000

[7] J. Rambough, I. Jacobson, G. Booch, *The Unified Modeling Language Reference Manual*, Addison-Wesly, 1999