Stanisław Jerzy Niepostyn[1]

Koncepcja architektury sterowanej regułami spójności modeli (CMDA) na przykładzie systemu e-IRZ dla ARiMR

1. Wstęp

Utworzenie jednego, prostego i zrozumiałego diagramu opisującego wszystkie aspekty projektowanego oprogramowania nie jest możliwe. Właściwym rozwiązaniem okazało się przedstawienie architektury oprogramowania za pomocą powiązanych modeli opisujących wybrane aspekty systemu informatycznego. Przedstawianie tego samego systemu z różnych perspektyw może jednak prowadzić do powstania niespójności pomiędzy modelami. Do budowy modeli opisujących oprogramowanie przyjęto powszechne stosowanie języka UML[2] (*Unified Modeling Language*). Z użyciem UML za pomocą różnorodnych diagramów można opisać system informatyczny z różnych perspektyw. Semantyka UML jest opisana w języku naturalnym i nie zawiera cech języka formalnego, co znacznie ogranicza możliwości kontroli spójności i kompletności diagramów UML, jak również opisu całego systemu informatycznego zawartego w tych diagramach. Uzyskanie takiej spójności opisu wymaga kontroli spójności modeli przedstawionych na diagramach UML.

Powyższe braki standardu UML powodują m.in., iż budowa dużych i skomplikowanych systemów IT jest rozpoczynana jedynie na podstawie wstępnego zarysu architektury oprogramowania (np. tylko przypadki użycia bądź tylko lista procesów biznesowych, a nawet sam opis tekstowy). Właściciele systemów informacyjnych, którzy chcą wdrożyć (zbudować) system IT, otrzymują na etapie wyboru wykonawcy praktycznie jedynie zatomizowane dane o przewidywanym koszcie i czasie trwania budowy oraz wdrożenia systemów IT, bez informacji o projekcie budowy oprogramowania (architektury oprogramowania).

[1] Politechnika Warszawska, Wydział Elektroniki i Technik Informacyjnych.

[2] http://www.omg.org (data odczytu: 22.11.2015).

Architektura oprogramowania systemów informatycznych jest dokumentowana najczęściej dopiero po ich zbudowaniu (wdrożeniu), przy czym taki opis zazwyczaj jest niespójny i niekompletny. Efektem powyższych praktyk są nagminne opóźnienia w realizacji budowy systemów IT, ich niedoszacowanie, a także brak spójnej i kompletnej dokumentacji, co prowadzi do braku możliwości zmiany wykonawcy systemu IT lub powierzenia rozbudowy istniejącego systemu IT innej firmie.

W artykule zaproponowano metodę opisu architektury oprogramowania na podstawie reguł spójności i kompletności uporządkowanego szeregu modeli UML od najbardziej abstrakcyjnego w postaci diagramu kontekstowego do modelu wykonywalnego (umożliwiających m.in. bezpośrednio wygenerowanie oprogramowania), którą w dalszej części będzie nazwana CMDA (ang. *Consistent Model Driven Architecture*).

Zaprezentowanie sposobu budowy systemu informatycznego za pomocą koncepcji CMDA prowadzi do: określenia możliwości wdrożenia projektowanego systemu IT, bezpieczeństwa wytworzenia oprogramowania, automatyzacji, uzasadnienia i predykcji czasu wyprodukowania systemu IT, a przede wszystkim oceny wiarygodności konstruowanej architektury oprogramowania.

Warto zaznaczyć, iż zaprojektowanie architektury systemu IT zgodnie z proponowaną koncepcją CMDA znacznie zmniejszy ryzyko niepowodzenia wdrożenia tego systemu w porównaniu z realizacją projektów IT, w których architektura oprogramowania jest opracowywana dopiero w trakcie ich realizacji, co jest zgodne z doświadczeniami autora zdobytymi w trakcie realizacji dużych projektów IT w administracji publicznej.

W pracy przedstawiono koncepcję kontroli spójności modeli UML na podstawie modelu perspektyw architektonicznych „4 + 1" P. Kruchtena[3] z zastosowaniem środowiska *Functionality-Structure-Behaviour*[4] J.S. Gero z 1990 r. Definicje spójności pokazano w części 2 niniejszego artykułu. W kolejnej części przedstawiono koncepcję CMDA, będącą oryginalnym rozwiązaniem autorskim, wraz z przykładami zastosowania w projekcie systemu informatycznego e-IRZ dla Agencji Restrukturyzacji i Modernizacji Rolnictwa w ramach zamówienia publicznego na doradztwo IT.

---

[3] P. Kruchten, *Architectural Blueprints – The "4+1" View Model of Software Architecture*, „IEEE Software" 1995, vol. 12(6), November, s. 42–50.

[4] J.S. Gero, *Design prototypes: a knowledge representation schema for design*, „AI Magazine" 1990, vol. 11(4), s. 26–36.

2. Spójność modeli – stan badań

Problemem spójności modeli opisujących system informatyczny zaczęto się szczególnie interesować w końcu lat 80. ubiegłego wieku. Formalny opis zagadnienia niespójności modeli zaproponowali G. Spanoudakis i A. Zisman[5] w 2001 r. Zauważyli oni, że niespójności powstają pomiędzy modelami opisującymi ten sam system z różnych punktów widzenia i używających wspólnych elementów. Wspólny element może być inaczej interpretowany w różnych modelach. W 2002 r. J. Hausmann[6] zaproponował definicję spójności modelu przypadków użycia, 3 lata później D. Berardi[7] podał definicję spójności diagramu klas, a w 2008 r. S. Jurack[8] przedstawił definicję spójności diagramu aktywności. Przywołane powyżej definicje należy traktować jako uzupełnienie reguł budowy modeli opisanych w standardzie UML.

Należy zauważyć, iż – zgodnie z koncepcją środowiska *Functionality-Structure-Behaviour* J.S. Gero – definicja J. Hausmanna dotyczy funkcjonalności, definicja D. Berardiego struktury, a definicja S. Juracka behawioryzmu docelowego systemu. W dalszej części artykułu powyższe cechy nazwano wymiarami architektury oprogramowania.

Wraz z rozwojem standardu UML zarysowały się pewne trendy w obszarze badań nad spójnością modeli UML, jak podano w artykule I. Ha[9] z 2008 r.: metody oparte na metamodelu (*meta-model based*), scenariuszach (*scenario-based*), ograniczeniach (*constraint-based*), metody grafowe (*graph-based*) czy oparte na wiedzy (*knowledge-based*). Niniejsza koncepcja wykorzystuje przede wszystkim metodę opartą na ograniczeniach. Jednakże są również używane reguły spójności zidentyfikowane w innych obszarach badań nad spójnością modeli UML. Z tego względu w dalszej części pracy zaproponowano przedstawianie

---

[5] G. Spanoudakis, A. Zisman, *Inconsistency management in software engineering: survey and open research issue*s, w: *Handbook of Software Engineering and Knowledge Engineering*, red. S.K. Chang, World Scientific Publishing Co., Singapore 2001, s. 329–380.

[6] J. Hausmann, R. Heckel, G. Taentzer, *Detection of Conflicting Functional Requirements in a Use Case-Driven Approach*, Proceedings of International Conference on Software Engineering, 2002.

[7] D. Berardi, A. Cali, D. Calvanese, G. Di Giacomo, *Reasoning on UML class diagrams*, „Artificial Intelligence Journal" 2005, vol. 168, issue 1, October, s. 70–118.

[8] S. Jurack, L. Lambers, K. Mehner, G. Taentzer, *Sufficient Criteria for Consistent Behavior Modeling with Refined Activity Diagrams, Model Driven Engineering Languages and Systems*, „Lecture Notes in Computer Science" 2008, vol. 5301, s. 341–355.

[9] I. Ha, B. Kang, *Cross Checking Rules to Improve Consistency between UML Static Diagram and Dynamic Diagram*, „Lecture Notes in Computer Science" 2008, vol. 5326, s. 436–443.

reguł spójności w postaci wyrażeń regularnych. W ogólnych regułach spójności zastosowano duże litery alfabetu jako symbole tzw. diagramów projektowych UML (diagramy projektowe to w większości diagramy UML, ale mające szczególne zastosowanie w wybranych etapach modelowania architektury oprogramowania). W opisie szczegółowych reguł spójności zastosowano dodatkowo małe litery alfabetu jako symbole elementów występujących na diagramach UML. Tak przedstawione reguły spójności umożliwiają pokazanie poszczególnych odwzorowań elementów w różnych modelach architektury oprogramowania systemów IT, których to mapowań zwykle nie pokazuje się w dokumentacji systemów informatycznych.

Tabela 1. Ogólne reguły spójności

Autor, rok	Sekwencja diagramów	Liczba reguł spójności
Egyed 2000	P(CQ|CO|CS)	50
Sapna 2007	C(S|U(A|Q))	18
Ibrahim 2012	UQC	8
Ha 2008	O(Q|A|I)	7
Chanda 2009	UAC	4
Hausmann 2002	UAO	3

Źródło: opracowanie własne na podstawie: A.F. Egyed, *Heterogeneous View Integration and its Automation*, PhD diss. in University of Southern California, 2000; P.G. Sapna, H. Mohanty, *Ensuring Consistency in Relational Repository of UML Models*, 10th International Conference on Information Technology, 2007, s. 217–222; N. Ibrahim, R. Ibrahim, M.Z. Saringat, D. Mansor, T. Herawan, *Use case driven based rules in ensuring consistency of UML model*, „AWERProcedia Information Technology & Computer Science" 2012, vol. 1, s. 1485–1491; J. Chanda, A. Kanjilal, S. Sengupta, S. Bhattacharya, *Traceability of Requirements and Consistency Verification of UML Use Case, Activity and Class diagram: A Formal Approach*, International Conference on Methods and Models in Computer Science, 2009, s. 1–4; J. Hausmann, R. Heckel, G. Taentzer, *Detection of Conflicting Functional Requirements in a Use Case-Driven Approach*, Proceedings of International Conference on Software Engineering, 2002.

W tabeli 1 ujęto zidentyfikowane przez poszczególnych autorów reguły spójności, a przedstawione w niniejszej pracy za pomocą wspomnianych wyżej wyrażeń regularnych, przy czym poczyniono następujące oznaczenia dla diagramów projektowych: A – diagram realizacji biznesowych przypadków użycia, B – diagram biznesowych przypadków użycia, C – diagram klas biznesowych, D – diagram wdrożenia, I – diagram komunikacji, J – diagram klas systemowych, L – diagram obiektów systemowych, M – diagram komponentów, O – diagram obiektów biznesowych, P – diagram pakietów, Q – diagram realizacji wewnątrzsystemowych przypadków użycia, R – diagram dekompozycji procesów, S – diagram stanów instancji klas biznesowych, T – diagram stanów instancji klas systemowych, U – diagram systemowych przypadków użycia, X – diagram

kontekstowy, Y – diagram wewnątrzsystemowych przypadków użycia, Z – diagram systemowych przypadków użycia.

W tabeli 2 przedstawiono wybrane szczegółowe reguły spójności w obszarze badań nad regułami spójności, przy czym zaproponowano następujące oznaczenia dla elementów diagramów: a – aktor, c – klasa, e – zdarzenie, f – atrybut, h – metoda, i – instancja, l – *lifeline*, m – komunikat, o – *occurence*, p – partycja, s – stan, t – przejście stanów, u – przypadek użycia, v – czynność. W celu zwiększenia przejrzystości reguł spójności do niektórych oznaczeń dodano stereotypy, np. e<<subevent>> oznacza podzdarzenie.

Tabela 2. Szczegółowe reguły spójności

Autor, rok	Szczegółowa reguła spójności
Ibrahim 2012	ChQm
Sapna 2007	ChQm
Ha 2008	QmOh, OhIm
Chanda 2009	UeAvCh

Źródło: opracowanie własne na podstawie: N. Ibrahim, R. Ibrahim, M.Z. Saringat, D. Mansor, T. Herawan, *Use case driven based rules in ensuring consistency of UML model*, „AWERProcedia Information Technology & Computer Science" 2012, vol. 1, s. 1485–1491; P.G. Sapna, H. Mohanty, *Ensuring Consistency in Relational Repository of UML Models*, 10th International Conference on Information Technology, 2007, s. 217–222; I. Ha, B. Kang, *Cross Checking Rules to Improve Consistency between UML Static Diagram and Dynamic Diagram*, „Lecture Notes in Computer Science" 2008, vol. 5326, s. 436–443; J. Chanda, A. Kanjilal, S. Sengupta, S. Bhattacharya, *Traceability of Requirements and Consistency Verification of UML Use Case, Activity and Class diagram: A Formal Approach*, International Conference on Methods and Models in Computer Science, 2009, s. 1–4.

3. Koncepcja CMDA

Budowa systemów informatycznych z wykorzystaniem koncepcji CMDA polega na sukcesywnych przekształceniach elementów od diagramu najbardziej abstrakcyjnego do diagramów implementowalnych w trzech perspektywach architektury oprogramowania: biznesowej (opis z punktu widzenia procesów biznesowych), systemowej (opis zewnętrzny systemu informatycznego) oraz komponentowej (opis wewnętrzny systemu informatycznego).

Poniżej zaprezentowano autorską definicję spójności na potrzeby wprowadzenia koncepcji CMDA.

definicja 1: modele są spójne, gdy spełniają warunek wystarczającej kompletności oraz warunek wystarczającej spójności (możliwość automatyzacji budowy systemu IT);

definicja 2 (warunek wystarczającej kompletności): modele systemu informatycznego powinny opisywać funkcjonalność, behawioryzm i strukturę projektowanego systemu (reguła kompletności) zgodnie z koncepcją środowiska *Functionality-Structre-Behaviour* J.S. Gero z 1990 r.;
definicja 3 (warunek wystarczającej spójności): transformacje (mapowanie) od diagramu na najwyższym poziomie abstrakcji[10] aż do wynikowego kodu aplikacji powinny spełniać definicje spójności:

a) między diagramami – według definicji Spanoudakisa i Zismana,
b) dla diagramu strukturalnego – według definicji Berarrdiego,
c) dla diagramu behawioralnego – według definicji Juracka,
d) dla diagramu funkcjonalności – według definicji Hausmanna.

W dalszej części artykułu zostaną przedstawione uporządkowane szeregi diagramów w postaci wyrażeń regularnych w celu zweryfikowania koncepcji CMDA wraz z przykładami systemu e-IRZ, którego projekt został opracowany przez autora w ramach zamówienia publicznego na doradztwo IT dla ARiMR.

Rysunek 1. Generyczny model uporządkowanego szeregu w koncepcji CMDA
Źródło: opracowanie własne.

[10] Na przykład diagram kontekstowy – S. Friedenthal, H. Lykins, A. Meilich, 2001 r.

Na rysunku 1 pokazano model generyczny uporządkowanego szeregu diagramów projektowych UML, który można zapisać w postaci wyrażenia regularnego:

$$X_{FSB}(R_{FSB}|B_F)\ A_{FSB}(C_S|S_B|U_F)Z_{FSB}(J_S|T_B|Y_F)Q_{FSB}M_{FSB}D_{FSB}. \quad (1)$$

Indeksy dolne w (1) oznaczają poszczególne wymiary architektury oprogramowania (F – funkcjonalność, B – behawioryzm, S – struktura).

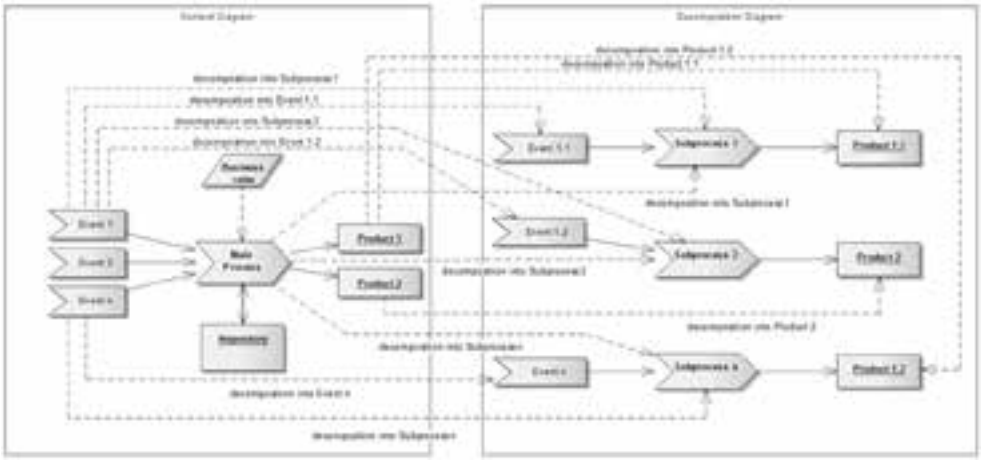
Na rysunku 2 zademonstrowano sposób uszczegółowienia budowy poszczególnych sekwencji diagramów z zaproponowanego szeregu (1). W dalszej części pracy pominięto reguły kompletności poszczególnych modeli, koncentrując się wyłącznie na regułach spójności zidentyfikowanych pomiędzy sąsiednimi diagramami.

Dla pierwszej sekwencji XR szeregu (1) zidentyfikowano następujące reguły spójności (rysunek 2):

R1) XeRe<<subevent>> – dekompozycja zdarzeń wejściowych na podzdarzenia,

R2) Xev<<process>>{1}R[v1<<subprocess>>-v2<<subprocess>>] – dekompozycja procesu głównego na podprocesy według zdarzeń wejściowych,

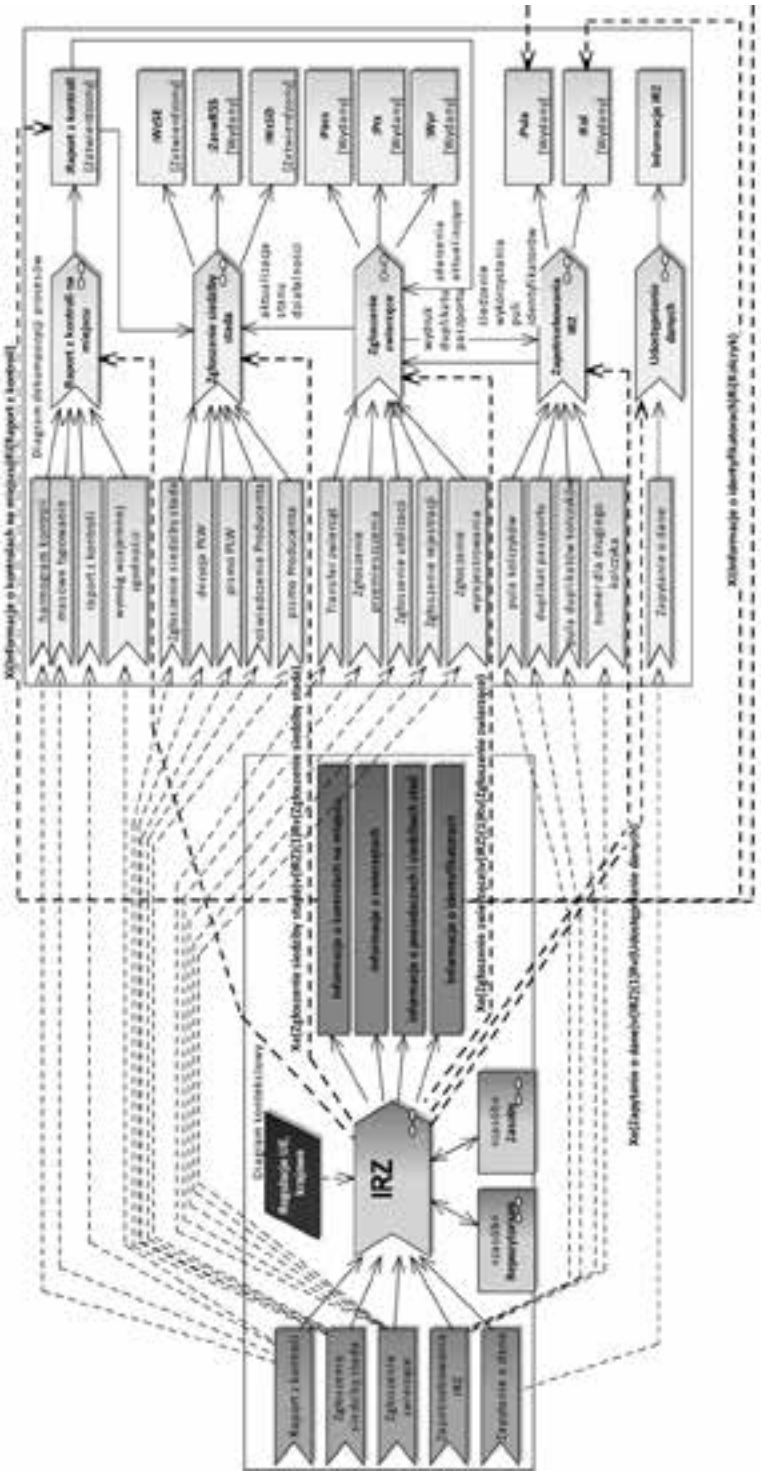
R3) Xi<<rules>>v<<process>>{1}i<<product>>Ri<<subproduct> – dekompozycja produktów wyjściowych na podprodukty wyjściowe.



Rysunek 2. Sekwencja XR – identyfikowanie reguły spójności

Źródło: opracowanie własne.

Dla modelu systemu e-IRZ, jak na rysunku 3, zastosowano wszystkie trzy reguły spójności – od R1 do R3.

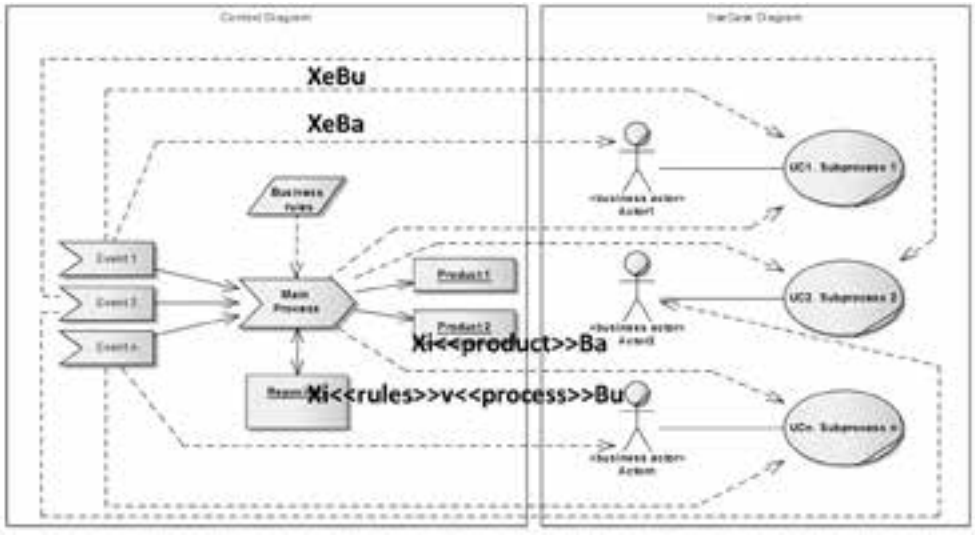


Rysunek 3. Sekwencja XR dla systemu e-IRZ

Źródło: opracowanie własne.

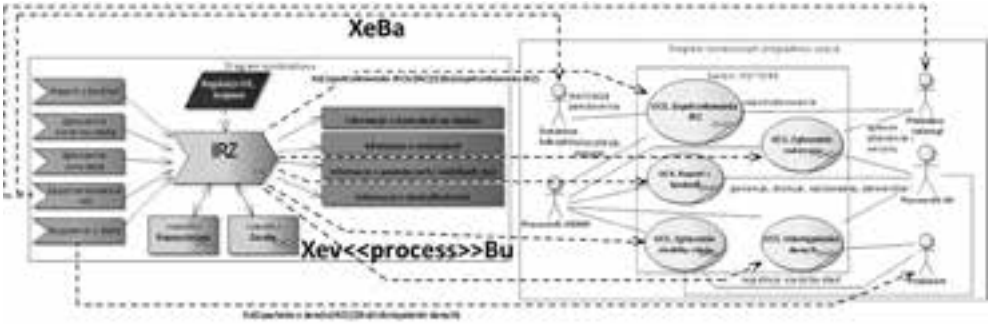
Dla kolejnego szeregu XB zidentyfikowano następujące reguły spójności (rysunek 4):

R4) XeBu – mapowanie zdarzeń na biznesowe przypadki użycia,
R5) XeBa – mapowanie zdarzeń na aktorów,
R6) Xi<<product>>Ba – mapowanie produktów na aktorów,
R7) Xi<<rules>>v<<process>>{1}Bu – mapowanie procesu na przypadki użycia.



Rysunek 4. Sekwencja XB – identyfikowanie reguły spójności

Źródło: opracowanie własne.



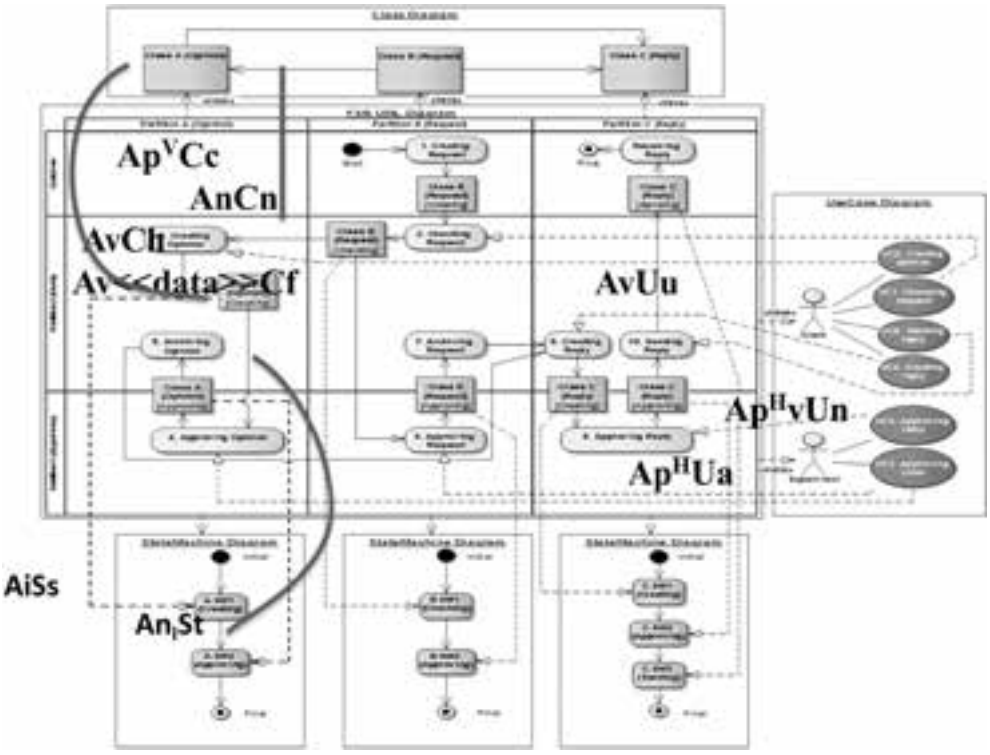
Rysunek 5. Sekwencja XB dla systemu e-IRZ

Źródło: opracowanie własne.

Dla modelu systemu e-IRZ, jak na rysunku 5, zastosowano reguły spójności R4, R5 oraz R7.

Postępując podobnie w przypadku kolejnych sekwencji diagramów szeregu (1), można zidentyfikować przykładowe reguły spójności dla sekwencji A_{FSB} ($C_S|S_B|U_F$) (rysunek 6):

R8) ApVCc – mapowanie partycji pionowej na klasy,
R9) AnCn – mapowanie związków danych na związki między klasami,
R10) AvCh – mapowanie czynności na metody klas,
R11) Av<<data>>Cf – mapowanie czynności na atrybuty klas,
R12) AvUu – mapowanie czynności na przypadki użycia,
R13) ApHUa – mapowanie partycji poziomych na aktorów,
R14) ApHvUn – mapowanie przyporządkowania czynności do partycji na związki pomiędzy przypadkami użycia a aktorami,
R15) AiSTATESs – mapowanie instancji oznaczonych stanem na stany,
R16) AnSt – mapowanie związków danych na tranzycje między stanami.



Rysunek 6. Sekwencja A_{FSB} ($C_S|S_B|U_F$) – identyfikowanie reguły spójności

Źródło: opracowanie własne.

Przykładowo, dla modelu systemu e-IRZ opisanego w załączniku „Model architektury systemu e-IRZ" materiałów przetargowych w zamówieniu publicznym ogłoszonym przez Agencję Restrukturyzacji i Modernizacji Rolnictwa na budowę, wdrożenie, utrzymanie i rozwój systemu e-IRZ[11] (m.in. diagram BPMN z rozdziału 2.1.1 i diagram UC z rozdziału 4.2) dla sekwencji A (diagram BPMN)U(diagram UC) zastosowano reguły spójności R12-R14.

Identyfikując kolejne reguły spójności (i kompletności) w powyższy sposób, w pozostałych sekwencjach diagramów szeregu (1) otrzymano zastosowane reguły spójności w poszczególnych modelach opracowanego modelu architektury oprogramowania systemu e-IRZ jak w tabeli 3.

Tabela 3. Szczegółowe reguły spójności zastosowane w modelu architektury oprogramowania projektowanego systemu e-IRZ

Perspektywa biznesowa	Perspektywa systemowa	Perspektywa komponentowa
Wymiar struktury		
XeRe<<subevent>>Ap^{V}Cc	ZiJc	QlMqDw
Xi<<product>>Ri<<subproduct>>Ap^{V}Cc	ZiJc	QlMqDw
Xvi<<product>>Rvi<<subproduct>>AnCn	ZnJn	QmMyDn
Xev<<process>>Rv<<subprocess>>AvCh	ZvJh	QmMyDn
Xev<<process>>BnAp^{H}vCh	ZvJh	QmMyDn
Xvi<<repository>>Rv<<data>>Av<<data>>Cf	Zv<<data>>Jf	QmMyDn
Xv<<data>>Bu<<data>>Av<<data>>Cf	Zv<<data>>Jf	QmMyDn
Wymiar funkcjonalności		
Xei<<rules>>v<<process>>BuAvUu	ZvYu	QlMqDw
Xei<<rules>>i<<product>>BaAp^{H}Ua	Zp^{V}Ya	Ql$_1$MyDn
Xev<<process>>BnApHvUn	Zp^{V}vYn	Ql$_1$MyDn
Wymiar behawioryzmu		
Xei<<product>>Rei<<subproduct>>Ai^{STATE}Ss	ZiTs	QoMyDn
Xvi<<product>>Rvi<<subproduct>>AnSt	ZnTt	QmMyDn
XeReAp^{V}St	ZnTt	QmMyDn
Xi<<product>>Ri<<subproduct>>Ap^{V}St	ZnTt	QmMyDn

[11] http://www.arimr.gov.pl/o-arimr/przetargi/przetargi/artykuly/dzp-2618–12015.html (data odczytu: 22.11.2015).

4. Podsumowanie i kierunki dalszych badań

W niniejszej pracy przedstawiono koncepcję *Consistent Model Driven Architecture*, za pomocą której można:

- zaprezentować planowaną metodykę budowy oprogramowania;
- ocenić wiarygodność projektowanej architektury oprogramowania;
- zmniejszyć ryzyko niepowodzenia wdrożenia systemu IT;
- uzyskać spójną i kompletną dokumentację systemu IT aktualną w całym cyklu życia projektu.

W dotychczasowej praktyce autora zastosowanie opisanej koncepcji CMDA znacznie wzbogaciło, przyśpieszyło i zautomatyzowało wiele prac związanych z projektowaniem oraz budową aplikacji internetowych opartych na metodach obiektowych. Koncepcja ta czy wybrane jej sekwencje były zastosowane przy analizie i projektowaniu systemu ECS/ICS dla Ministerstwa Finansów, a także przy analizie i projektowaniu systemu informatycznego dla RWE Polska. Koncepcję CMDA w pełni zastosowano przy projektowaniu systemu ZSIN (zintegrowany system informacji o nieruchomościach) oraz systemu e-IRZ (system identyfikacji i rejestracji zwierząt) dla Agencji Restrukturyzacji i Modernizacji Rolnictwa.

Obecnie kontynuowane są prace badawcze na podstawie badań przemysłowych przy opracowywaniu narzędzia do generowania poszczególnych modeli systemów IT z modelu generycznego CMDA.

Bibliografia

Berardi D., Cali A., Calvanese D., Di Giacomo G., *Reasoning on UML class diagrams*, „Artificial Intelligence Journal" 2005, vol. 168, issue 1, October, s. 70–118.

Chanda J., Kanjilal A., Sengupta S., Bhattacharya S., *Traceability of Requirements and Consistency Verification of UML UseCase, Activity and Class diagram: A Formal Approach*, International Conference on Methods and Models in Computer Science, 2009.

Egyed A.F., *Heterogeneous View Integration and its Automation*, PhD diss. in University of Southern California, 2000.

Gero J.S., *Design prototypes: a knowledge representation schema for design*, „AI Magazine" 1990, vol. 11(4), s. 26–36.

Ha I., Kang B., *Cross Checking Rules to Improve Consistency between UML Static Diagram and Dynamic Diagram*, „Lecture Notes of Computer Science" 2008, vol. 5326, s. 436–443.

Hausmann J., Heckel R., Taentzer G., *Detection of Conflicting Functional Requirements in a Use Case-Driven Approach*, Proceedings of International Conference on Software Engineering, 2002.

Ibrahim N., Ibrahim R., Saringat M.Z., Mansor D., Herawan T., *Use case driven based rules in ensuring consistency of UML model*, „AWERProcedia Information Technology & Computer Science" 2012, vol. 1, s. 1485–1491.

Jurack S., Lambers L., Mehner K., Taentzer G., *Sufficient Criteria for Consistent Behavior Modeling with Refined Activity* Diagrams, Model Driven Engineering Languages and Systems, „Lecture Notes in Computer Science" 2008, vol. 5301, s. 341–355.

Kruchten P., *Architectural Blueprints – The "4+1" View Model of Software Architecture*, „IEEE Software" 1995, vol. 12(6), November, s. 42–50.

Sapna P.G., Mohanty H., *Ensuring Consistency in Relational Repository of UML Models*, 10th International Conference on Information Technology, 2007.

Spanoudakis G., Zisman A., *Inconsistency management in software engineering: survey and open research issues*, w: *Handbook of Software Engineering and Knowledge Engineering*, red. S.K. Chang, World Scientific Publishing Co., Singapore 2001, s. 329–380.

Źródła sieciowe

http://www.arimr.gov.pl/o-arimr/przetargi/przetargi/artykuly/dzp-2618–12015.html (data odczytu: 22.11.2015).

http://www.omg.org (data odczytu: 22.11.2015).

* * *

Consistency Model Driven Architecture – the consistency rules of UML models in software architecture

Summary

The CMDA concept supports the design of an IT system from a top level abstract context model diagram towards implementable models that enable us to generate target software based on rules of consistency between the elements of specific models. The design regime presented leads to the derivation of a set of chosen and consistent models that fully and completely describes the software architecture. The CMDA concept has already been positively validated in several industry projects, among others, for example, the design of the Animals Identification and Registration System (e-IRZ), and the Business Process Management platform, both for the Agency for Restructuring and Modernisation of Agriculture.

Keywords: consistency rules, completeness rules, UML, UML diagram series