

Ewolucja architektury – implikacje dla systemów bazodanowych

prof. dr hab. inż. Czesław Jędrzejek

*Centrum Doskonałości w dziedzinie Telematyki
Instytut Automatyki i Inżynierii Informatycznej
Politechnika Poznańska*

Streszczenie:

Ostatnie lata przyniosły duży rozwój metod tworzenia systemów i programowania obiektowego. Należy tu wymienić standaryzację metamodeli danych, CWM, oraz np. programowanie aspektowe, AOP. CWM pozwala transformować dane ze źródeł o strukturze relacyjnej, hierarchicznej, sieciowej czy opartej na XML. Powstaje pytanie czy twórcy systemów wykorzystujących relacyjne bazy danych powinni przejmować się zagadnieniami takimi jak dziedziczenie, enkapsulacja, polimorfizm czy refleksja i diagramy UML. Narzędzia do rozwoju systemów sterowanych modelami (Model Driven Development) w sposób bardzo istotny wspierają ewolucję architektury, czy tworzenie rodzin produktowych i są bardziej wszechstronne niż systemy CASE w aspekcie tworzenia projektu systemu. Dla dużych projektów może się opłacać stosować modelowanie obiektowe i wykorzystując profile danych transformować modele UML do modeli relacyjnych. Niniejszy referat prezentuje przegląd wymienionych zagadnień technicznych oraz podaje w sposób popularny przykłady ich zastosowania. Stosowanie technik obiektowych pozwala tworzyć systemy bardziej odporne na zmiany oraz refaktoryzować kod. Zilustrowane będą funkcjonalności nakładek na standardowe narzędzia Oracle 10g Designer i Oracle JDeveloper 10g.

Informacja o autorze:

W początkowym okresie pracy związany z AGH i UJ w Krakowie. Przez okres 10 lat odbywał staże naukowe i pracował jako Visiting Professor kolejno na kilku uczelniach w USA. W roku 1994 związał się Francusko-Polską Wyższą Szkołą Nowych Technik Informatyczno-Komunikacyjnych (EFP) w Poznaniu. W latach 1999-2004 zajmował stanowisko Wiceprezesa Zarządu firmy ITTI w Poznaniu. Jest autorem lub współautorem ponad 100 publikacji. Kierował kilkudziesięcioma projektami dla wiodących operatorów oraz dostawców sprzętu telekomunikacyjnego w Polsce w zakresie ewolucji sieci i usług, inżynierii ruchu w sieciach teleinformatycznych oraz wykonania, integracji i wdrożenia systemów informatycznych. Od 2003 r. zajmuje stanowisko profesora w Instytucie Automatyki i Inżynierii Informatycznej Politechniki Poznańskiej. Jest koordynatorem Centrum Doskonałości w dziedzinie TELEMATYKI w Politechnice Poznańskiej. Realizował kilka projektów europejskich dotyczących aplikacji informatycznych.

1. Wstęp

W dziedzinie systemów i aplikacji zorientowanych na dane istnieje dychotomia. Relacyjne systemy zarządzania bazami danych zdominowały rynek i wyparły obiektowe systemy zarządzania bazami danych. Firmy będące liderami w dziedzinie obiektowych systemów zarządzania bazami danych (Object-Oriented Database Management Systems, ODBMS) działają w dziedzinach niszowych. Przykładami są firmy: Gemstone [1], Objectivity [2], Versant [3] (BMW Car IT GmbH wdraża ODBMS FastObjects [4]) i Computer Associates (System Jasmine) [5], które posiadają ważnych klientów. Inne firmy zostały wchłonięte: Ardent przez Ascential [6] a następnie IBM; firma ObjectStore została przemianowana na Progress Software [7].

Organizacja Object Data Management Group opracowała w roku 2001 standard dla ODBMS (“The Object Data Standard: ODMG 3.0”) [8] po czym się rozwiązała, co jest symptomatyczne. Standard ten obejmuje m.in.

- Model Danych,
- Język Opisu Danych (DDL, Data Description Language),
- Język Manipulacji Danymi (DML, Data Manipulation Language),
- Język Kwerend (QL, Query Language), [9] i
- Wiązania m.in. do C++, a wiązanie do Javy zostało przejęte przez Java Data Objects, JDO.

Niestety model danych w ODMG 3.0 nie jest optymalny (dyskusja na wysokim poziomie formalnym znajduje się w [10]). Z kolei UML 2.0 nie posiada systemu składowania danych.

Niniejszy artykuł nie zajmuje się rynkową akceptowalnością ODBMS lub obiektowo-relacyjnych systemów zarządzania bazami danych.

Tematem artykułu jest ewolucja architektury [11]. Mamy z nią do czynienia w dwu przypadkach:

1. w czasie tworzenia systemu
2. przy tworzeniu tzw. rodzin produktowych. Należy sobie zdawać sprawę, że np. w dziedzinie elektroniki użytkowej, jeśli nawet sprzedaje się 1 mln sztuk urządzenia, może ono występować w kilkudziesięciu modelach lub wersjach.

Jednym z podejść do ewolucji architektury są Software Factories - modele dziedzinowe (np. banku elektronicznego), z których przy pomocy języków dziedzinowych (DSL, Domain-Specific Languages) można generować aplikacje [12].

Inną metodą jest ewolucja architektury wykorzystująca rozwój sterowany modelami (ang. Model-Driven Development, MDD), który jest rozszerzeniem paradygmatu architektury sterowanej modelami, Model-Driven Architecture (MDA) na wszystkie aspekty rozwoju systemu.

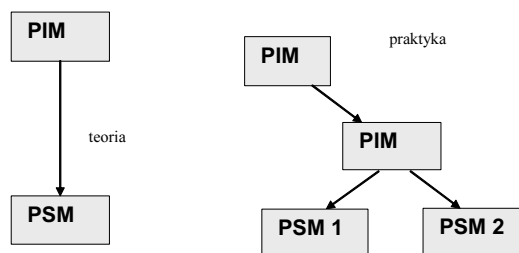
2. MDA

MDA (użycie słowa „architektura” w nazwie jest mylące) sprowadza się do [13]:

1. przeniesienia ciężaru rozwoju systemu na wyższy poziom abstrakcji i nadanie modelowaniu centralnej roli,
2. ścisłe oddzielenie warstw systemu,
3. automatyczna generacja kodu z modelu.

Ogólnie można wyróżnić cztery warstwy oprogramowania

- **Computation Independent Model (CIM)** – domenowy model biznesowy, nie pozostający w ścisłej relacji z technologią informatyczną,
- **Platform Independent Model (PIM)** – abstrakcyjna specyfikacja systemu wykorzystująca metamodel,
- **Platform Specific Model (PSM)** – model odwzorowany na konkretne rozwiązania wybranej platformy,
- **Implementation Model** – kod niskopoziomowy (np. Java, C++).



Rys. 1. Relacje między artefaktami w warstwach

W praktyce przejście z PIM do PSM jest bardziej złożone niż przedstawiono po lewej stronie Rys. 1.

Architektura sterowana modelami jest zbiorem standardów, obecnie w wersji MDA 1.0 rozwijanych przez OMG (Object Management Group) [14]. Składa się ona z następujących specyfikacji: Unified Modeling Language (UML), Meta-Object Facility (MOF), XML Meta-Data Interchange (XMI) oraz Common Warehouse Meta-model (CWM). Jest coraz częściej stosowana w przypadku projektów „od początku”, Podniesienie poziomu abstrakcji opisu systemu osiąga się poprzez wprowadzenie metamodelu. Narzędzia MDA umożliwiają zachowanie wzajemnych związków pomiędzy modelami, oraz umożliwienie transformacji modeli na podstawie ustalonych wcześniej reguł i standardów.

3. Metamodelę

3.1. Notacja MDA (MOF)

Organizacja OMG zaproponowała język modelowania UML jako język do opisu wszystkich rodzajów programowych artefaktów podejścia obiektowego. Aby umożliwić definiowanie podobnych języków, przeznaczonych dla innych celów, opracowano język wyższego poziomu, stanowiący notację określaną jako MOF (*Meta-Object Facility*). UML i MOF stanowią podstawę czteropoziomowego stosu modeli metodyki MDA (Tab. 1).

Tabela 1. Czteropoziomowa organizacja procesu modelowania w metodyce MDA

Poziomy	Terminy MOF	Przykłady	Pojęcia	Typowe użycie
M3	meta-metamodel	Model „MOF”	Klasy MOF	Twórca standardów
M2	metamodel, meta-metadata (metadane)	metamodel UML, metamodel CWM	Klasy UML Podstawowe pojęcia: klasa, asocjacje, typ danych, stałe	Twórca narzędzi

Poziomy	Terminy MOF	Przykłady	Pojęcia	Typowe użycie
M1	model, metadane	modele UML, metadane CWM	Klasa: samochód	Twórca aplikacji
M0	obiekt, dane	Modelowane systemy, Dane w hurtowni	Instancja: BMW	Użytkownik

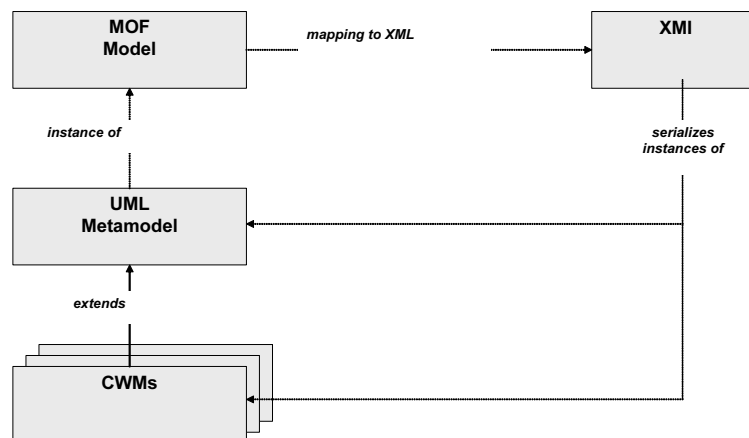
Na poziomie M0 występują konkretne przypadki (realizacje) modeli, na przykład konkretne wykonanie programu w *C*, lub rekord danych opisujących konkretną osobę. Na poziomie M1 występują modele, na przykład program w *C*, lub typ rekordu dla osoby. Na poziomie M2 mamy do czynienia z metamodelami, które odgrywają rolę gramatyk do definiowania modeli np. pojęcia relacyjnego modelu danych.

W końcu, na poziomie M3 występuje meta-metamodel, W standardzie MDA, MOF umożliwia definiowanie wielu meta-modeli na poziomie M2. Takimi zaakceptowanymi meta-modelami są obecnie UML i CWM.

W metodyce MDA przyjęto:

- Unified Modeling Language, **UML** jako standardowy język do definiowania metamodeli,
- XML Meta-Data Interchange **XMI** jako standardowy mechanizm wymiany metadanych i metamodeli za pomocą XML,
- Common Warehouse Meta-model **CWM** jako standardowa specyfikacja do definiowania struktury i semantyki metadanych w hurtowni danych i eksploracji danych,
- **MOF to IDL Mapping** jako standardowy mechanizm dostępu do metadanych za pomocą API (niezależnie od języka programowania i modelu obiektowego),
- **MOF to Java Mapping** jako standardowy mechanizm dostępu do metadanych w Javie.

Wprowadzenie rekomendacji XMI pozwoliło na serializację metamodeli - reprezentację diagramów w formacie XML.

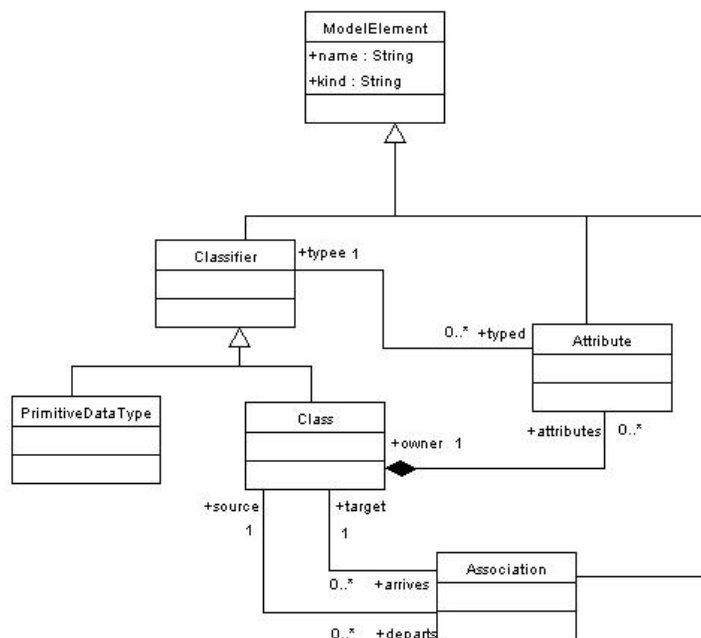


Rys. 2. Serializacja metamodeli za pomocą XMI (w formacie XML)

Wykorzystanie XMI pozwala - w szczególności - automatycznie przetwarzać modele kompatybilne z MOF i opisane za pomocą XMI, w taki sposób jak przetwarzane są dokumenty XML-owe za pomocą XSLT lub XQuery. Obecnie najnowsza wersja standardu OMG to XML Metadata Interchange (XMI) v2.0, z maja 2003 r., która wprowadza schematy XML. Niestety OMG ma duże

kłopoty z utrzymaniem interoperacyjności. MDA. I tak np. CWM 1.1 jest tak jak CWM 1.0 oparty na UML 1.3, MOF 1.3, XMI 1.1. Niestety w momencie standaryzacji CWM używano DTD, a nie schematy XML.

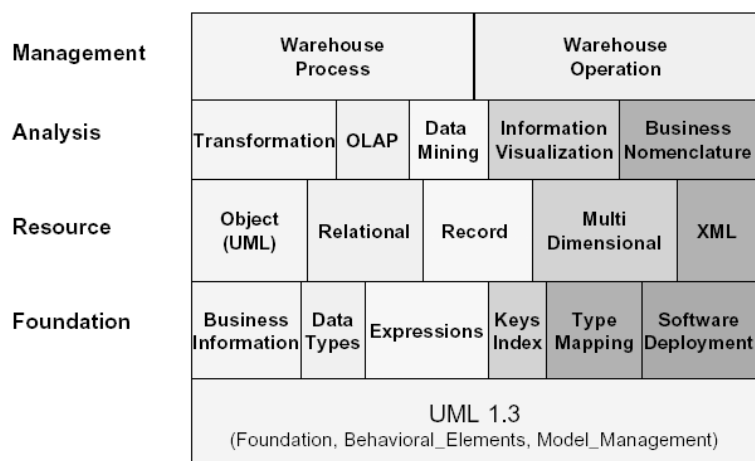
Rys. 3 przedstawia metamodel UML 2.0.



Rys. 3. Metamodel UML

3.2. CWM – Common Warehouse Metamodel

W procesach projektowania, budowy i zarządzania systemami informatycznymi dużą wagę przywiązuje się do zarządzania danymi metamodelu. CWM definiuje metamodel (model modelu danych, Rys. 4) przeznaczony do opisu i zarządzania metadanymi, związanymi z hurtowniami danych. Każda metaklasa w CWM dziedziczy z pewnej metaklasz należącej do Object Model. Na przykład w CWM Relational Package, metamodel Relational definiuje metaklasę Table, która reprezentuje dowolną tabelę relacyjnej bazy danych. Ta metaklasa wyprowadzona jest z metaklasz Class należącej do Object Model. Podobnie metaklasa Column jest instancją metaklasz Attribute. W ten sposób określona jest semantyczna zależność między pojęciami „klasa” i „tabela” oraz „atrybut” i „kolumna”. Pełny metamodel CWM jest przedstawiony na Rys. 4.

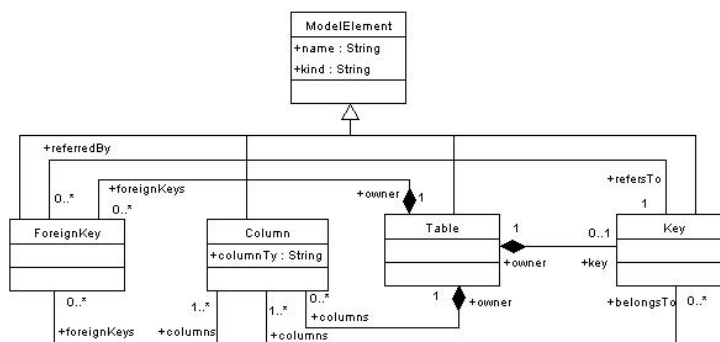


Rys. 4. Metamodel CWM

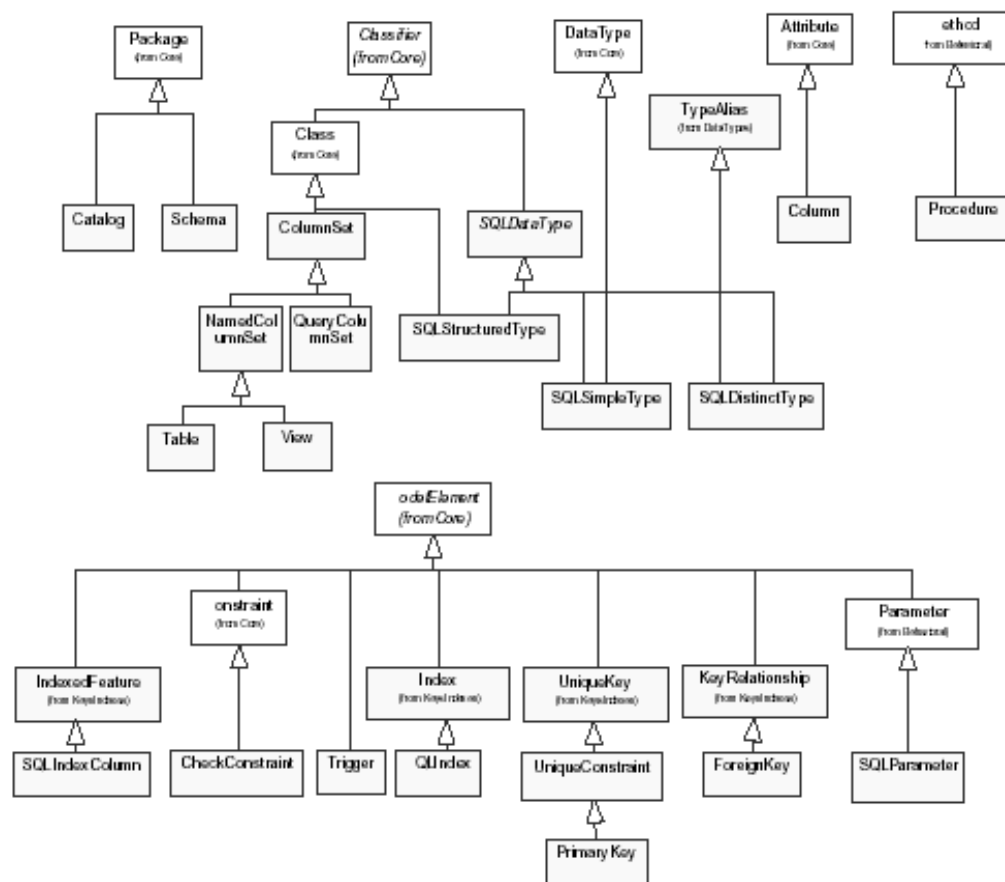
CWM zawiera model obiektów – Object Model, który oparty jest na UML. Składa się z wersji metamodelu UML, która nie dotyczy hurtowni danych. Ten model obiektowy służy dwóm celom:

- jako baza, na której zbudowany jest metamodel CWM,
- jako metamodel obiektowych zasobów danych.

Na Rys. 5 przedstawiono fragment modelu relacyjnego z asocjacjami, a na Rys. 7 pełny (ale bez asocjacji), pakiet zasobu (*resource*) modelu relacyjnego, występujący w metamodelu CWM.



Rys. 5. Model klas fragmentu relacyjnej struktury danych

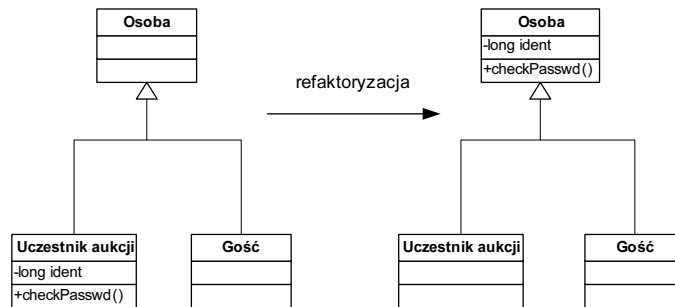


Rys. 6. Pakiet modelu relacyjnego (dziedziczenie)

4. Ewolucja Architektury

Często mamy do czynienia z sytuacją zmiany systemu informatycznego, np. zmiany struktury danych. Metody na zwiększanie odporności kodu na zmiany w systemach relacyjnych zostały przedstawione w [14]. Jedną z metod ułatwiających dodanie funkcjonalności do kodu jest refaktoryzacja [15] – zmiana konstrukcji kodu bez modyfikowania jego działania. Skutkiem refaktoryzacji kodu jest jego ulepszenie, np. w celu zastosowania w nim wzorców projektowych. Wprowadzenie wzorców projektowych do kodu znacznie ułatwia jego późniejsze modyfikacje i ewentualne rozbudowy. Przykład restrukturyzacji hierarchii dziedziczenia jest przedstawiony na Rys 7.

4.1. Przykłady ewolucji kodu

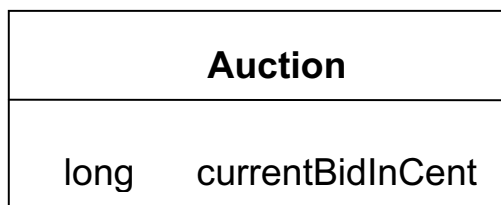


Rys. 7. Przeniesienie atrybutów i metod do supekasy

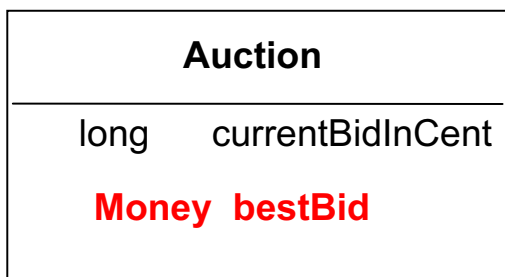
Przedstawię teraz procedurę zmiany struktury danych wraz z czynnościami programistycznymi.

1. Identyfikacja starej struktury danych

Chcemy zastąpić : long przez Money



2. Dodanie nowej struktury danych i kompilacja programu



3. Identyfikacja niezmiennika (inwariant) – zawsze w programie wywołanie bestBid valueInCent() powinno zwracać wartość zmiennej currentBidInCent

```
currentBidInCent ==
    bestBid.valueInCent()
```

4. Dodanie nowej struktury danych i niezmienników (przy zmianie starej struktury). Dodatkowo należy wykonać kompilację i testy. assert (warunek) generuje komunikat jeśli warunek nie jest spełniony.

```
currentBidInCent = ...
    bestBid.setValue...
    assert ( currentBidInCent == bestBid.valueInCent())
```

5. Modyfikacja miejsc, gdzie była użyta stara struktura

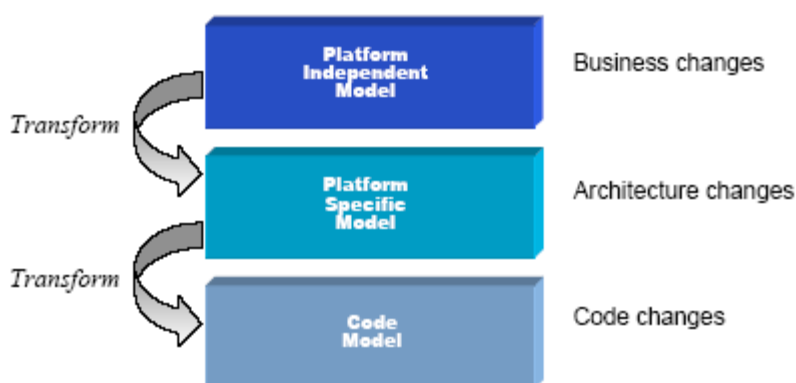
```

= ... currentBidInCent ...
-----
  ↓
= ... bestBid.valueInCent()

```

6. Usunięcie starej struktury danych .

Ewolucję architektury można ułatwić operując na metamodelach. W tym celu należy jest zdefiniować język transformacji, służący odwzorowaniu tych metamodeli, jak przedstawiono na Rys. 8.



Rys. 8. Transformacje modeli

Do tego niezbędne jest standaryzacja języka transformacji QVT (*Query, View, Transformation*) czym obecnie zajmuje się organizacja OMG.

4.2. Pozycja firmy Oracle odnośnie MDA

Jak każda wielka firma informatyczna Oracle posiada oficjalne stanowisko odnośnie metamodeli i MDA Oracle. Warehouse Builder używa metadanych w oparciu o Common Warehouse Model (CWM 1.0), którego Oracle był jednym z twórców. W teorii metadane CWM powinny być przenaszalne do narzędzi innych firm. W praktyce zgodność jest ograniczona. OWB10g importuje dane z narzędzi Erwin firmy Computer Associates oraz Powerdesigner firmy Powersoft. Począwszy od OWB 9.2 przy pomocy adaptera „MITI Bridges” firmy Metaintegration Technology metadane mogą być wymieniane z Cognos ReportNet, Microsoft Visio i Business Objects.

Oracle systematycznie rozbudowuje narzędzia do modelowania na poziomie PSM zezwalające na deklaratywną, graficzną modyfikację kodu. Narzędzia te dostarczają profili i wzorców projektowych w zakresie :

1. Java modeler
2. EJB modeler
3. Business Component modeler
4. Database modeler
5. Web Services modeler

6. XML modeler

7. Struts modeler

Od Oracle JDeveloper 10g release 10.1.3 możliwości zostały poszerzone o BPEL process integration modeler. Oracle posiada szerokie związki partnerskie z firmami dostarczającymi bogatsze funkcjonalności na poziomie PIM lub dodatkowych transformacji [17]. W Tab. 2 przedstawiono wybrane produkty partnerskie.

Tabela 2. Funkcjonalności wybranych narzędzi współpracujących z produktami Oracle

Kategoria	Firma	Produkt	Opis
UML	Softimeam	<u>Objecteering/UML</u>	Narzędzie UML o kompleksowej funkcjonalności (analiza wymagań, zapewnienie spójności, generacja kodu) oraz profile UML
DB-Java	Orinda Software Ltd	<u>OrindaBuild</u>	Tworzy DAO Factory i klasy usług dla procedur PL/SQL i deklaracji SQL
UML	CanyonBlue	<u>konesa</u>	Pełna funkcjonalność UML Modeling dla JDeveloper 10g
O/R Mapping	Visual Paradigm	<u>DB Visual Architect</u>	Object Relational Mapping Tool and DB Modeling
Forms to JDeveloper	Oracle Consulting	<u>Oracle JHeadstart</u>	Migruje Oracle Forms applications do aplikacji ADF i/lub generowanie w pełni funkcjonalnych aplikacji ADF z Oracle Designer.
Refaktoryzacja	Aqris Software	<u>Refactorit</u>	Narzędzie do refaktoryzacji dla Oracle JDeveloper 10g

6. Podsumowanie

Wydaje się, że została już przekroczona masa krytyczna rozwoju MDA. Narzędzia wspomagające MDA są szczególnie przydatne dla celu ewolucji architektury. Powstaje coraz więcej narzędzi wspierających standard i są bardziej wszechstronne niż systemy CASE w aspekcie tworzenia projektu systemu oraz wsparcia rozwoju systemu w całym cyklu życia. Dla dużych projektów może się opłacać stosować modelowanie obiektowe i wykorzystując profile danych transformować modele UML do modeli relacyjnych

Jednocześnie nie należy utożsamiać zarządzania danymi z systemami zarządzania bazami danych. Bazy danych współpracują z aplikacjami tworzonymi w językach obiektowych np. Java lub C++. Liderem w dziedzinie narzędzi MDA jest IBM z platformą eclipse. W oparciu o to otwarte narzędzie powstają wtyczki do coraz większej liczby narzędzi. Microsoft nie zaakceptował ani MOF, ani UML decydując się na wzorce projektowe i języki dziedzinowe (DSL). Strategia Oracle jest ostrożna, ale elementy MDA są systematycznie wdrażane, nie rezygnują z tradycyjnych linii narzędzi.

Bibliografia

1. <http://www.gemstone.com/>
2. <http://www.objectivity.com/>
3. www.versant.com/;
4. www.jdocentral.com/pdf/JavaSpektrum_BMWcarIT_engl.pdf

5. www3.ca.com/Solutions/Product.asp?ID=3008
6. <http://www.ascential.com/>,
7. <http://www.progress.com/realtime/company/index.ssp>
8. R. Cattell, D. Barry: *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann 2000.
9. Subieta K.: *Teoria i konstrukcja obiektowych języków zapytań*, Wydawnictwo PJWSTK, Warszawa 2004, ISBN 83-89244-28-4
10. Habela P, Subieta K.: *Overcoming the Complexity of Object-Oriented DBMS Metadata Management*. D.Konstantas, M.Léonard, Y.Pigneur, S.Patel (Eds.): *Object-Oriented Information Systems, 9th International Conference, OOIS 2003, Geneva, Switzerland, September 2-5, 2003, Proceedings*. Lecture Notes in Computer Science 2817 Springer 2003, ISBN 3-540-40860-6: s. 214-225
11. R. Lopez-Herrejon, D. Batory, W. Cook, *Evaluating Support for Features in Advanced Modularization Technologies*, ECOOP 2005 Programming Languages and Operating Systems LNCS 3586, 169-194,
12. <http://lab.msdn.microsoft.com/teamsystem/workshop/sf/>
13. www.omg.org/mda/
14. J. Gnybek, *Sprytny PL/SQL*, Ploug'ka nr 34; <http://www.ploug.org.pl/plougunki.php?action=read&p=33&a=2>
15. M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Professional, 1999.
16. *UML Modeling and MDA in Oracle JDeveloper 10g*, August 2004; www.oracle.com/technology/products/jdev/collateral/papers/10g/uml_mda_sod.pdf
17. www.oracle.com/technology/products/jdev/htdocs/partners/index.html