STUDIA INFORMATICA 2009
Volume 30 Number 2A (83)

Mateusz LECHMAN, Andrzej GRUDZIEŃ, Tomasz TRACZYK
Politechnika Warszawska, Instytut Automatyki i Informatyki Stosowanej

ZASTOSOWANIE JĘZYKA XVCL DO BUDOWY REPOZYTORIUM DIAGRAMÓW KLAS

Streszczenie. Artykuł opisuje koncepcję zarządzania ewolucją diagramów klas z wykorzystaniem wersjowania opartego na atrybutach języka XCVL – dialektu XML, służącego do zarządzania wariantami i programowania generatywnego. Repozytorium zbudowano jako trójwarstwową hierarchię ramek XVCL, tworzącą dość uniwersalną strukturę, która może również zostać użyta do zarządzania zmiennością innych artefaktów projektowych.

Słowa kluczowe: ewolucja systemów, UML, programowanie generatywne, XCVL, zarządzanie wersjami

APPLYING XVCL TO BUILDING CLASS DIAGRAM REPOSITORY

Summary. The paper describes the idea of class diagram evolution management with attribute-driven versioning and XVCL language – the XML dialect for variant description and generative programming. The proposed repository is designed as a three layer hierarchy of XVCL frames. This universal structure can also be used to manage changes of other project artifacts.

Keywords: information system evolution, UML, generative programming, XVCL, version management

1. Wprowadzenie

Systemy informacyjne mają zwykle długi czas eksploatacji i w tym czasie stale ewoluują. Nie jest rzadką sytuacja, gdy ten sam system musi mieć wersje działające na kilku różnych platformach. Wreszcie także dość często zdarza się, iż systemy przeznaczone dla kilku klientów różnią się tylko szczegółami, a niekiedy początkowo są identyczne, a z czasem – w miarę ewolucji – pojawiają się różnice. To wszystko powoduje, że utrzymanie systemów jest trud-

ne, między innymi z powodu braku odpowiednich narzędzi, zdolnych do zarządzania kolejnymi wersjami i równoległymi wariantami systemów.

W sformalizowanych metodach tworzenia oprogramowania kluczową rolę odgrywają modele, m. in. modele struktur danych. Jedną z podstawowych technik modelowania używanych współcześnie są UML-owe diagramy klas. Diagramy te powinny być utrzymywane w czasie rozwoju systemu, by odzwierciedlać aktualny stan projektu. Jeśli jednak projekt rozwija się w wielu równoległych wariantach, utrzymanie aktualnych diagramów jest bardzo trudne.

Praca ta stanowi próbę rozwiązania tego problemu, wykorzystującą programowanie generatywne i zaawansowany model wersjowania sterowanego atrybutami.

Technologia programowania generatywnego, której przedstawicielem jest język XVCL, wydaje się być szczególnie dobrze dopasowana do zarządzania równoległymi wariantami oprogramowania. Nie zawiera ona jednak rozwiązań związanych z ewolucją. Dlatego do możliwości tej technologii dodać trzeba było metodę zarządzania wersjami, wybraną w taki sposób, by jak najlepiej odzwierciedlić naturę zmienności systemów informacyjnych: zmiany zwykle mają przyczyny należące do wielu niezależnych od siebie i występujących jednocześnie grup.

Przedstawiona praca stanowi kontynuację badań nad możliwościami użycia języka XVCL do zarządzania ewolucją systemów informacyjnych, prezentowanych m. in. w [1, 2, 3, 4, 5].

2. Zastosowane techniki i komponenty

2.1. Model wersjowania

Gdy rozważamy problem zarządzania zmianami dowolnych obiektów, jednym z pierwszych zagadnień do rozstrzygnięcia jest wybór modelu wersjowania. W niniejszej pracy zdecydowano się na wybór modelu opartego na atrybutach [10].

W modelu tym każda wersja (czy wariant) opisana jest przez zbiór par atrybut – wartość. Wybranym atrybutom mogą zostać przypisane taksonomie: główna, wiążąca wszystkie dopuszczalne wartości dla atrybutu, oraz dodatkowe, używające podzbioru dopuszczalnych wartości oraz mogące korzystać z wartości innych atrybutów [2]. Podejście to pozwala na modelowanie różnych relacji między wersjami (np. następstwa czy generalizacji), według których można lokalizować wersje i warianty w repozytorium. Dodatkową zaletą jest zawarcie informacji o cechach wersji czy wariantu w atrybutach ją opisujących.

Przed dokonaniem ostatecznego wyboru omówiony model został porównany z innymi możliwościami. Najprostszy z modeli – szeregowy – został odrzucony, ponieważ nie uwzględnia istnienia wariantów, których użycie jest niezbędne, np. w przypadku dostarczania produktu dwóm klientom o nieznacznie różniących się wymaganiach. Kolejnym rozważanym modelem był model równoległy, spotykany w wielu popularnych narzędziach [11]. Przyczyną jego odrzucenia jest znacząca trudność wprowadzania modyfikacji w środowisku wielowariantowym. Dla przykładu, jeżeli wymagamy, aby utrzymywać warianty różniące się docelową platformą (np. Java i .NET) oraz pewnymi wymaganiami klienta, to w modelu równoległym oznacza to utworzenie osobnego wariantu dla każdej kombinacji platforma – klient. W takiej sytuacji wspomniana trudność modyfikacji pojawia się, gdy próbujemy wprowadzić zmianę czy poprawkę dotyczącą jednego zagadnienia, np. konkretnej platformy. Konieczne jest wtedy wprowadzenie jej osobno w każdym wariancie związanym z daną technologią, a różniącym się docelowym klientem.

Przy dokonywaniu wyboru brano również pod uwagę modele bardziej skomplikowane, takie jak wersjowanie zorientowane na zmiany [12], model oparty na środowisku wersji [13] czy wersjowanie zorientowane obiektowo [14]. Względem modelu opartego na atrybutach nie wnoszą jednak one dostatecznie dużo, aby zniwelować znacznie wyższy poziom komplikacji – stąd decyzja o rezygnacji z ich użycia.

2.2. Języki XCVL i XQuery

Technologia i język XVCL [6] należą do rozwiązań generatywnych. Oparte są na idei budowy systemu jako zbioru adaptowalnych komponentów oraz zasad ich łączenia. Podstawowymi celami XVCL są: eliminowanie redundancji na różnych poziomach abstrakcji (od powielonego pomysłu projektowego do redundancji kodu) oraz ułatwienie uwzględniania zmian w systemie. XVCL jako język jest dialektem XML przypominającym prosty język proceduralny.

Oprócz XVCL w projekcie zastosowano też język XQuery [7]. Jego wprowadzenie podyktowane było niewygodą użycia XVCL do realizacji zapytań do repozytorium wg modelu opartego na atrybutach, w szczególności do operacji na grafach. XQuery jest językiem deklaratywnym przeznaczonym do przeszukiwania i przekształcania dokumentów XML, co – po wypracowaniu prostego dialektu zapisu taksonomii w postaci XML – pozwoliło na swobodne użycie XQuery w realizacji repozytorium.

Dodatkowo język XQuery może zostać wykorzystany do wprowadzania do repozytorium artefaktów projektowych zapisywanych w postaci XML, tworzonych w zewnętrznych narzędziach typu CASE.

2.3. Wykorzystania środowiska Topcased do prezentacji artefaktów projektowych

Tworzenie i modyfikowanie artefaktów projektowych powinno się odbywać w przeznaczonych dla nich narzędziach CASE. W tym celu zaproponowano integrację tworzonego repozytorium ze środowiskiem Topcased (ang. *Toolkit In OPen source for Critical Applications & SystEms Development*).

Celem projektu Topcased jest wytworzenie takiego narzędzia *open source* do modelowania systemów, które przez wysoką konfigurowalność umożliwi szybką adaptację zmian w stosowanych metodologiach. Środowisko to dostarcza możliwość modelowania w warstwie[1] M3. Pozwala więc zdefiniować własny język, co może zostać wykorzystane do zaprojektowania specyficznych graficznych edytorów np. porównujących warianty diagramów. Tworzenie takich edytorów odbywa się w sposób deklaratywny na podstawie zdefiniowanego w ECORE metamodelu i plików konfiguracyjnych, które określają odpowiadający mu interfejs użytkownika. Natywnym sposobem zapisu diagramów w Topcased jest XMI [15].

3. Architektura rozwiązania

3.1. Wcześniejsze prace na temat wersjowania diagramów z wykorzystaniem XVCL

W pracy [3] opisana jest próba zastosowania języka XVCL do zapisu ewolucji diagramów dla struktur bazy danych. Jako wersjowany artefakt przyjęto tam zapis XMI diagramu klas generowany przez narzędzie *Rational Rose*.

W strukturze rozwiązania wyróżniono następujące części:

- zbiór ramek generatora – odpowiadających za przekształcenie informacji o diagramie zawartych w repozytorium do postaci dokumentu XMI,
- ramki opisu elementów diagramu – zawierają szczegółowy opis elementów, takie jak definicje kolumn dla encji czy krotności dla związków,
- ramka konfiguracyjna – określa, które elementy (encje i związki) wchodzą w skład diagramu, a także specyfikacje różnic.

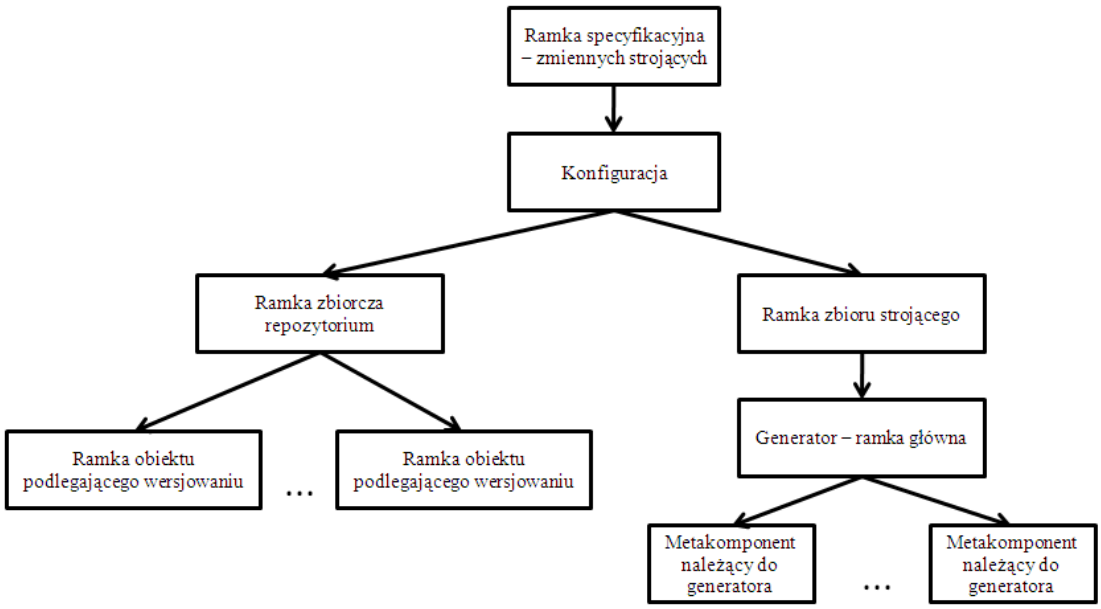
Obiekty są wersjowane według modelu zorientowanego na zmiany: wskazanie zmian dotyczących danej wersji/wariantu znajduje się w ramce konfiguracyjnej, natomiast szczegóły zmiany znajdują się w samej ramce obiektu. Rozwiązanie to pozwala na łatwe uzyskanie informacji o zmianach tak poszczególnych składowych, jak i diagramu jako całości.

[1] Konsorcjum OMG (*Object Management Group*) definiuje cztery poziomy abstrakcji dla procesu modelowania: M0 – rzeczywisty obiekt (np. kod klasy języka Java), M1 – model (diagram klas), M2 – metamodel (np. metamodel UML), M3 – meta-metamodel (języki pozwalające zapisać obiekty z warstwy M2 i zależności między nimi w sposób strukturalizowany, np. EMOF, CMOF, ECORE).

Choć rezultaty prac były dość obiecujące, natrafiono także na spore przeszkody, które z czasem spowodowały konieczność zmiany kierunku poszukiwań. Pierwsze przeszkody miały charakter czysto techniczny: stosowane w [3], cieszące się wówczas sporą renomą narzędzie *Rational Rose* dawało wyjątkowo nieporządny kod XMI, w którym np. przy każdym zapisie modelu zmieniały się wartości identyfikatorów obiektów, co niezmiernie utrudniało utrzymanie repozytorium. W dodatku, wkrótce producent zaprzestał rozwoju tego narzędzia, wskutek czego straciło ono znaczenie i pozycję rynkową. Ponadto, okazało się, że użyty wówczas prosty model wersjowania nie odzwierciedla w pełni rzeczywistych potrzeb projektowych. Dlatego prace przeorientowano, wybierając bogatszy model zarządzania wersjami: wersjowanie oparte na atrybutach, oraz narzędzie, które wprawdzie nie jest równie renomowane, ale ma dobrą opinię, w szczególności znane jest z wyjątkowo dobrej zgodności ze standardami.

Pierwsza próba zastosowania nowo wybranego modelu wersjowania dotyczyła repozytorium dla artefaktów warstwy ORM systemu [2]. W związku ze zmianą modelu wersjowania rozbudowana została struktura repozytorium do postaci zaprezentowanej na rys. 1. Kluczowe elementy zaprezentowane na rysunku to:

- ramka zbiorcza repozytorium – zawiera spis wszystkich obiektów w repozytorium i odpowiada za ich przetworzenie,
- ramka obiektu podlegającego wersjowaniu – zawiera opis cech wszystkich wersji i wariantów danego obiektu,
- generator – zbiór ramek tworzących artefakty dla konkretnego narzędzia ORM na podstawie danych zebranych z repozytorium,
- ramka zbioru strojącego – ramka pozwalająca na modyfikacje działania generatora,
- konfiguracja – ramka zawierająca zapis zapytania wg modelu opartego na atrybutach, wskazanie ramki generatora i ramki zbioru strojącego; informacja ta jest wystarczająca, by wygenerować konkretne artefakty,
- ramka specyfikacyjna – zawiera zbiór zmiennych sterujących procesem przetwarzania pozostałych ramek; jej zawartość jest tworzona automatycznie (z wykorzystaniem XQuery) na podstawie ramki konfiguracyjnej i taksonomii atrybutów użytych do wersjowania.

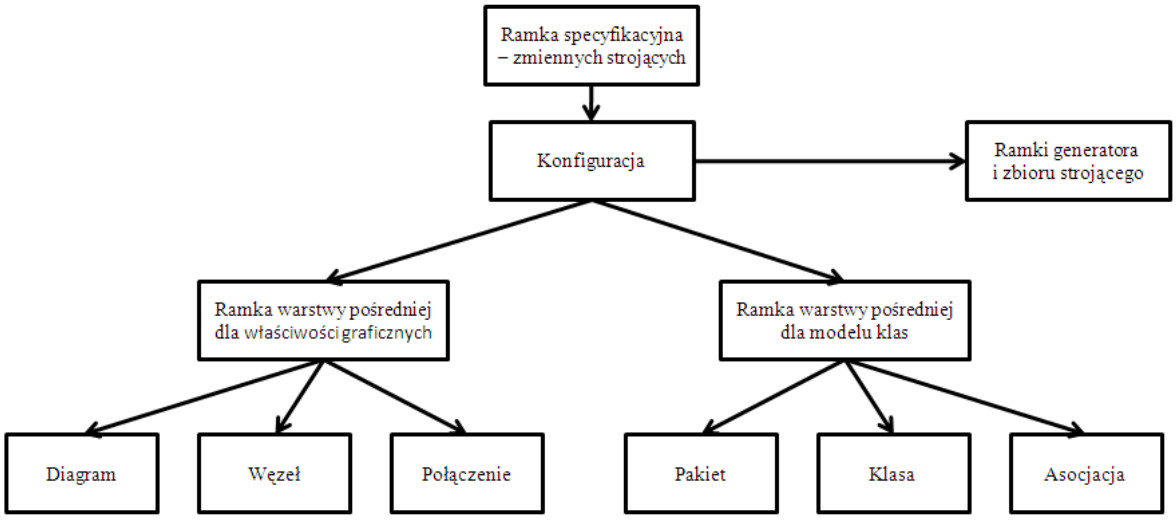


Rys. 1. Struktura adaptacji ramek w repozytorium warstwy ORM
Fig. 1. The structure of frames adaptation in the ORM layer repository

3.2. Modyfikacje wprowadzone w repozytorium diagramów klas

Jako punkt wyjścia przy pracach nad repozytorium diagramów klas przyjęto hierarchię ramek zaprojektowaną do zarządzania warstwą ORM. Podjęto próby przeorganizowania kodu źródłowego w celu wyodrębnienia informacji dotyczących diagramów klas od instrukcji języka XVCL, sterujących przetwarzaniem tej informacji, tak by była ona czytelna dla użytkownika korzystającego jedynie z edytora tekstu. Zaproponowano, aby adaptacja ramek, przechowujących wersjowane informacje o diagramach, odbywała się przez dodatkową warstwę pośrednią ramek XVCL. Zadaniem ramek z tej warstwy jest adaptacja wszystkich plików, w których przechowywane są informacje o obiektach, spełniających kryteria wyrażone przez użytkownika w ramce konfiguracji. Pobrane dane są umieszczane w odpowiednich przestrzeniach nazw zmiennych XVCL, te z kolei sterują generacją kodu.

Na rys. 2 przedstawiono strukturę adaptacji ramek w repozytorium diagramów klas. Z punktu widzenia procesu generacji kodu możemy wyróżnić w niej trzy główne warstwy: danych, ramek pośrednich (pobierających i przetwarzających dane) oraz generatorów. Warstwa danych składa się z modelu klas (ramki pakietów, klas i asocjacji) oraz właściwości graficznych (ramki diagramów, węzłów i połączeń).



Rys. 2. Struktura adaptacji ramek w repozytorium diagramów klas
Fig. 2. The structure of frames adaptation in the class diagram repository

Uproszczono również sposób oznaczania wersji z zastosowanego w pracy [2] (opartego na instrukcji `<select option>`) na konstrukcję wykorzystującą znacznik `<ifdef>`.

Ramki obiektów posiadających nazwy mające semantykę (a więc klasy, pakiety, asocjacje) są nazywane w postaci: `<nazwa_obiektu> (<unikalny_identyfikator_obiektu>)`. Na nazwy pozostałych składają się jedynie unikalne identyfikatory. Znaczniki `<set>` i `<set-multi>` wykorzystano do zapisu cech obiektu.

Poniżej umieszczono przykład zapisu klasy, która występuje w dwóch wariantach.

```
<x-frame name="Korespondencja (_0lP_ODbfEd2sDrQAZsMPuw)">
  <set var="id" value="_0lP_ODbfEd2sDrQAZsMPuw"/>
  <ifdef var="VERSION:Operating-system=Win2003;DB-type=Oracle">
       <set var="name" value="Korespondencja" />
       <set var="isAbstract" value="false" />
       <set var="visibility" value="public" />
       <set-multi var="properties" value="" />
       <!-- inne zmienne opisujące cechy klasy-->
    </ifdef>
  <ifdef var="VERSION:Operating-system=Win2003;DB-type=DB2">
       <set var="name" value="Korespondencja" />
       <set var="isAbstract" value="false" />
       <set var="visibility" value="public" />
       <set-multi var="properties" value="Adres" />
       <!--inne zmienne opisujące cechy klasy-->
    </ifdef>
</x-frame>
```

Zaletą zaproponowanego podejścia jest to, że ramki z danymi mogą być teraz łatwiej przeszukiwane przez użytkownika oraz przez zewnętrzne narzędzia, bez zważania na specyficzne instrukcje przetwarzające XVCL.

Adaptacja ramek przedstawionych na rysunku 2 została ustawiona na XVCL-owy typ *samelevel*, oznaczający, że zmienne zadeklarowane w ramkach adaptowanych są również widoczne w ramkach adaptujących. Powoduje to utworzenie dużej liczby parametrów steru-

jących generatorami na poziomie ramki konfiguracji. Aby poprawić przejrzystość kodu, wprowadzono przestrzenie nazw dla poszczególnych klas zmiennych przetwarzanych w repozytorium:

- `DATA:<Nazwa_przestrzeni_danych_używanej_przez_generator>:<id_obiektu>:<nazwa_parametru>` – zmienne zadeklarowane w ramkach warstwy pośredniej (przede wszystkich pochodzące z wczytanych plików danych),
- `CONFIGURATION:<Nazwa_parametru_z_ramki_konfiguracji>` – parametry określone w ramce konfiguracji,
- `TUNING_SET:<Nazwa_zbioru_strojącego>:<Nazwa_parametru>` – parametry ze zbiorów strojących,
- `VERSION:<Wersja>` – zmienne sterujące wyborem wersji.

Strukturę generatora diagramu klas zaprojektowano jako dwie hierarchie ramek zawierających fragmenty kodu XMI opatrzonego znakowaniem XCVL. Pierwsza z nich służy do generowania pliku modelu klas. Druga natomiast tworzy plik opisujący właściwości graficzne diagramu (sposób rozmieszczenia obiektów na diagramie, ich wielkości itd.). Wygenerowane obiekty są umieszczane w przestrzeni roboczej środowiska Topcased, co umożliwia natychmiastową analizę pobieranych diagramów w dedykowanych edytorach graficznych.

Opracowano także mechanizm wprowadzenia danych do repozytorium (ang. *check-in*) z wykorzystaniem XQuery, który pozwala zapisywać modyfikacje wykonane w Topcased. Narzędzie Topcased stanowi tym samym interfejs użytkownika, umożliwiający wygodne wprowadzenie zmian do diagramu klas pobranego z repozytorium. W zależności od doboru wartości atrybutów sterujących wersjowaniem zmodyfikowany diagram może zostać przypisany do dowolnej wersji systemu (w szczególności do nowego wydania lub wariantu).

Zestaw par atrybut-wartość, opisujący wersję diagramu klas dla operacji pobierania danych z repozytorium, definiuje się w ramce konfiguracji. Decyduje on, które obiekty i ich właściwości zostaną uwzględnione w wygenerowanym kodzie XMI. Z kolei, dla operacji wprowadzania diagramu do repozytorium wspomniany zestaw zostanie użyty do oznakowania właściwości jego wszystkich obiektów składowych.

3.3. Możliwości rozbudowy repozytorium

Konstrukcje zastosowane do opisu diagramów klas mogą zostać zastosowane do zarządzania zmiennością innych artefaktów projektowych. Zaproponowany zapis danych jest dość uniwersalny. Możemy wyróżnić w nim następujące generyczne elementy.

1. Wymiary – określają funkcjonalność repozytorium. Każdy wymiar definiuje pewną hierarchię obiektów. Aktualnie zaimplementowano dwa wymiary: model klas i właściwości

graficzne diagramu. Wymiary pozwalają na rozbudowę repozytorium o nowe funkcjonalności.

2. Klasyfikacje obiektów – pozwalają na tworzenie hierarchii obiektów. Umożliwia to m. in. traktowanie klas i asocjacji jako składowych pakietów i śledzenie zmienności takiej relacji. Klasyfikacje zaimplementowano przez strukturę katalogów (obiekty podrzędne składowane są w katalogu o nazwie obiektu nadrzędnego) i odwołania do nazw ramek opisujących obiekty podrzędne w obiektach nadrzędnych. Poniżej przedstawiono sposób zapisu klasyfikacji na przykładzie ramki pakietu:

```
<x-frame name="Pakiet 1 (Pakiet1ID)">
  <set var="id" value="Pakiet1ID"/>
  <ifdef var="VERSION:Operating-system=Win2003;DB-type=Oracle">
        <set var="name" value="Pakiet 1" />
        <set-multi var="containedClasses"
           value="AparatTelefoniczny (_0P2sGDb),Klient (_Slrdfia)”/>
        <set-multi var="containedPackages" value="" />
        <set-multi var="containedAssociations" value="kupuje (_G-Tbf)/>
   </ifdef>
</x-frame>
```

   Powyższy kod wskazuje (instrukcje `<set-multi>`), że w skład pakietu wchodzą dwie klasy i jedna asocjacja.

3. Obiekty – wyodrębnione elementy artefaktu projektowego, które podlegają wersjowaniu. W przypadku diagramów klas obiektami są pakiety, klasy, asocjacje, diagramy, węzły i połączenia. Każdy obiekt posiada unikalny identyfikator w ramach repozytorium.
4. Atrybuty – opisują właściwości obiektów.
5. Asocjacje – zapisanie odnośnika do innego obiektu przez wstawienie jego unikalnego identyfikatora do wartości atrybutu. Asocjacje stosowne są m. in. w obiektach typu węzeł do wskazania klas, które reprezentują na diagramie.

Dodanie nowej funkcjonalności do repozytorium (np. diagramów aktywności) sprowadza się do zaprojektowania nowego wymiaru (określenia obiektów, ich atrybutów i hierarchii). Należy następnie stworzyć skrypt XQuery, w którym definiuje się atrybuty obiektów i miejsce, skąd mają być pobierane. Fizyczne utworzenie i modyfikowanie ramek danych, składających się na nowy wymiar, obsługiwane jest przez opracowaną generyczną bibliotekę XQuery, która służy do zapisu do repozytorium obiektu dowolnego typu, poddanego wersjowaniu opartemu na atrybutach. Dodanie mechanizmu pobrania danych do zewnętrznego narzędzia sprowadza się do zadeklarowania reguł adaptacji hierarchii (w postaci ramki warstwy pośredniej) oraz utworzenia szkieletu kodu źródłowego.

Możliwości rozbudowy repozytorium sprawdzono już w praktyce, co potwierdziło korzystne cechy rozwiązania. Do repozytorium dodano mianowicie generator, tworzący specjalny diagram porównujący dwie wersje diagramów klas, co sprowadziło się do stworzenia hierarchii ramek nowego metakomponentu (wykorzystującej w dużej mierze już istniejący

kod) i wywołania jej w ramce konfiguracji. Pozostałe warstwy nie były w żaden sposób modyfikowane.

4. Podsumowanie

Zaprezentowane w referacie rozwiązanie może znaleźć zastosowanie do zarządzania ewolucją diagramów klas w przypadku systemów występujących w wielu wariantach i przeznaczonych do długoletniej eksploatacji. Dodatkowo, stanowi ono wzorzec projektowy do tworzenia innych repozytoriów – nie tylko diagramów UML – i ich integracji ze środowiskiem Topcased.

Zgrupowanie ramek XVCL w trzy dobrze zdefiniowane warstwy logiczne sprawia, że struktura stworzonego narzędzia jest czytelna i rozszerzalna. Funkcje repozytorium zrealizowano za pomocą deklaratywnych języków XVCL i XQuery, bez użycia języków proceduralnych. Przejrzysty kod tych języków i przejrzysta struktura danych stanowią dobry materiał do dalszego rozwijania zaprojektowanych diagramów, a także ułatwiają zrozumienie kodu i administrowanie nim.

Nie ma przeszkód, aby stworzone narzędzie mogło zostać włączone do ujednoliconego repozytorium bazującego na wersjowaniu opartym na atrybutach. Takie środowisko pozwoli generować zarówno kod, jak i powiązane z nim fragmenty dokumentacji (w tym diagramy), przez co w łatwy sposób będzie można śledzić wprowadzone zmiany i ich przyczyny we wszystkich warstwach artefaktów projektowych.

BIBLIOGRAFIA

1. Traczyk T.: Zarządzanie ewolucją systemu informacyjnego za pomocą programowania generatywnego i języka XVCL. II Krajowa Konferencja Naukowa Technologie Przetwarzania Danych, Poznań 2007.
2. Grudzień A., Traczyk T.: Zarządzanie wersjami warstwy ORM sterowane atrybutami. W ramach pracy zbiorowej pod redakcją S. Kozielskiego i in.: Bazy danych. Rozwój metod i technologii. Wydawnictwo Komunikacji i Łączności, 2008.
3. Bębenek A., Traczyk T.: Koncepcja użycia języka XVCL do zapisu ewolucji diagramów UML reprezentujących struktury baz danych. W ramach pracy zbiorowej pod redakcją S. Kozielskiego i in.: Bazy danych. Struktury, algorytmy, metody. Wydawnictwo Komunikacji i Łączności, 2006.

4. Grudzień A., Traczyk T., Jarzabek S.: Application of generative programming to evolution of object-relational mapping layer. The Second AIS SIGSAND European Symposium on Systems Analysis and Design, Sopot 2007.
5. Kowalski J., Traczyk T.: Zastosowanie języka XVCL do zarządzania ewolucją schematu relacyjnej bazy danych. W ramach pracy zbiorowej pod redakcją S. Kozielskiego i in.: Bazy Danych: Nowe Technologie. Wydawnictwo Komunikacji i Łączności, 2007.
6. XML-based Variant Configuration Language, http://xvcl.comp.nus.edu.sg
7. XML Query Language, http://www.w3.org/TR/xquery
8. The Open-source Toolkit for Critical Systems http://topcased.org
9. Object Management Group: Unified Modeling Language (UML), http://www.uml.org/uml
10. Gregic J.: Towards a versioning model for component-based software assembly. ICSM, 2003.
11. CVS – concurrent versions system v1.11.21. http://ximbiot.com/cvs/manual/cvs-1.11.21/cvs.html
12. Munch B. P.: Versioning in a Software Engineering Database - the Change Oriented Way. PhD thesis, The Norwegian Institute of Technology, 1995.
13. Wilkes W., Klahold P., Schlageter G.: A general model for version management in databases. In Wesley W. Chu, Georges Gardarin, Setsuo Ohsuga, and Yahiko Kambayashi, editors, VLDB'86 12th International Conference on Very Large Data Bases, August 25-28, 1986, Kyoto, Japan, Proceedings, Morgan Kaufmann, 1986, s. 319÷327.
14. Agrawal R., Buroff S., Gehani N. H., Shasha D.: Object versioning in Ode. In Proceedings of the IEEE 7th International Conference on Data Engineering, Kobe, Japan, 1991, s. 446÷455.
15. XML Metadata Interchange. http://www.omg.org/technology/documents/formal/xmi.htm

Recenzent: Dr inż. Piotr Bajerski

Wpłynęło do Redakcji 20 stycznia 2009 r.

Abstract

The paper describes an idea of class diagram evolution management with attribute-driven versioning and XVCL language – XML dialect for variant description and generative programming. The proposed repository is designed as a three layer hierarchy of XVCL frames: data, data access and code generators. This universal structure can also be used to manage changes of other project artifacts. The information from repository is processed with XVCL and XQuery languages, and the designed tool integrates with Topcased – highly configurable open source environment for systems modeling.

The frames structure used in earlier works for construction of the Object-Relational Mapping layer repository (Fig. 1) has been chosen as a beginning point for research. An additional class of frames has been introduced to separate information about versioning objects from XVCL instructions that control the adaptation process (Fig. 2). The proposed object-oriented data model handles components as hierarchies and associations, and can easily be extended. Generative programming technique is used to produce class diagrams in XMI (check-out functionality of repository), which can be modified in Topcased environment. Changes are introduced back to repository (check-in) through XQuery scripts.

The proposed solution will be used as a part of planned XVCL-based CASE tool, which will support development and maintenance of multi-variant information systems with databases.

Adresy

Mateusz LECHMAN, Politechnika Warszawska, Instytut Automatyki i Informatyki Stosowanej, ul. Nowowiejska 15/19, 00-665 Warszawa, Polska, M.Lechman@stud.elka.pw.edu.pl.
Andrzej GRUDZIEŃ, Politechnika Warszawska, Instytut Automatyki i Informatyki Stosowanej, ul. Nowowiejska 15/19, 00-665 Warszawa, Polska, A.Grudzien@elka.pw.edu.pl.
Tomasz TRACZYK: Politechnika Warszawska, Instytut Automatyki i Informatyki Stosowanej, ul. Nowowiejska 15/19, 00-665 Warszawa, Polska, T.Traczyk@ia.pw.edu.pl.