Zarządzanie ewolucją systemu informacyjnego za pomocą programowania generatywnego i języka XVCL

Tomasz Traczyk[1]

Streszczenie: Współczesne systemy informacyjne z bazami danych są złożonymi wytworami, zawierającymi wiele warstw i różnorodne oprogramowanie. Systemy te muszą przez długi czas odpowiadać ciągle zmieniającym się wymaganiom. Zarządzanie zmianami takich systemów jest szczególnie trudne, gdy występują one w wielu równolegle istniejących wariantach.

Referat opisuje koncepcję zarządzania ewolucją złożonych wielowariantowych systemów informacyjnych z bazami danych za pomocą programowania generatywnego i języka XVCL, prezentuje dotychczasowe badania oraz przedstawia dalsze zamierzenia.

Słowa kluczowe: ewolucja schematów, zarządzanie konfiguracją, programowanie generatywne (*generative programming*)

1. Wprowadzenie

1.1. Motywacja

Współczesne systemy informacyjne z bazami danych działają w warunkach ciągłej zmienności wymagań, wymuszanej nieustannymi zmianami otoczenia, której muszą sprostać w długim – zwykle przynajmniej kilkunastoletnim – okresie eksploatacji. Systemy te zwykle są bardzo złożone funkcjonalnie i technicznie: realizują setki funkcji, składają się z wielu warstw i zawierają różnorodne oprogramowanie. Zarządzanie zmianami takich systemów jest z reguły bardzo trudne, w szczególności zaś trudno jest osiągnąć taki stan, w którym przyczyny wszystkich wprowadzanych modyfikacji są klarownie zdefiniowane, a ich konsekwencje – łatwe do prześledzenia.

Zarządzanie zmianami staje się szczególnie trudne, gdy system istnieje równocześnie w wielu równoległych wariantach[2]. Warianty te mogą pojawiać się np. ze względu na (a) konieczność działania systemu na różnych platformach sprzętowych i/lub programowych, w szczególności na różnych DBMS; (b) zróżnicowane zakresy funkcjonalne i inne różnice między poszczególnymi wdrożeniami systemu, wynikające z różnic w wymaganiach poszczególnych użytkowników; (c) rozproszenie, gdy repliki fragmentów baz danych, znajdujące się w różnych węzłach, różnią się ze względu na specyficzne wymagania związane z danym węzłem.

Z punktu widzenia zarządzania zmianami tego typu systemów istotna jest możliwość przewidywania konsekwencji zmian przed ich wykonaniem, oraz prześledzenia

[1] Instytut Automatyki i Informatyki Stosowanej, Politechnika Warszawska, ul. Nowowiejska 15/19, 00-665 Warszawa
e-mail: T.Traczyk@elka.pw.edu.pl

[2] W tej pracy wersje systemu podzielimy – dla jasności – na *wydania* (*revisions*) następujące po sobie, związane z rozwojem systemu, oraz *warianty*, tj. wersje działające równocześnie (rys. 1).

– już po wdrożeniu zmiany – całej „ścieżki" od skutków zmiany do jej przyczyny. Zmiana może oczywiście dotyczyć równocześnie wielu wariantów systemu, przy czym jej oddziaływanie na poszczególne warianty może być różne; musi jednak być przewidywalne i dać się prześledzić także już po wprowadzeniu.

1.2. Zamierzenie badawcze

Podjęto próbę zbadania możliwości użycia programowania generatywnego i języka XVCL (patrz Jarzabek i in., 2003) jako środka do zarządzania zmianami w systemach informacyjnych mających wiele równoczesnych wariantów i zbudowanych w oparciu o współczesne technologie wykorzystujące bazy danych oraz nowoczesne środowiska budowania aplikacji. Badane są możliwości zastosowania języka XVCL do zarządzania zmianami poszczególnych elementów projektu takiego systemu i jego warstw. Kolejne części tego opracowania przedstawiają ideę programowania generatywnego z użyciem XCVCL, koncepcję jego zastosowania do opisanych wyżej celów, dotąd przeprowadzone prace i plany dalszych badań.

Celem całego zamierzenia jest opracowanie i zbudowanie prototypu systemu typu CASE, który – bazując na idei programowania generatywnego i na języku XVCL – umożliwiałby zarządzanie ewolucją złożonych wielowariantowych systemów z bazami danych.

Dorobek naukowy, dotyczący zarządzania ewolucją systemów informacyjnych, w szczególności zmianami struktur relacyjnych i obiektowych oraz wymiarów w systemach OLAP, jest bardzo bogaty. Przedstawiona tutaj koncepcja ma jednak ważne założenie, odmienne od tych, które leżą u podstaw większości uznanych prac w tym zakresie. Otóż zakłada się, że struktury danych, kod aplikacji, modele analityczne oraz dokumentacja są traktowane jako równorzędne artefakty, których zmienność powinna być zarządzana w ten sam sposób. Zastosowana notacja „generyczności" i sposoby adaptacji muszą być więc dość ogólne, a nie wyspecjalizowane pod kątem specyficznej części projektu czy rodzaju kodu (por. np. Lieberherr, 1995). Dlatego wyniki prac dotyczących ewolucji jednego tylko ze składników projektu mogą tu być wykorzystane w bardzo ograniczonym zakresie. Nasze podejście jest bliższe systemom zarządzania wersjami kodu źródłowego, w repozytorium projektu nie przechowuje się jednak gotowych wersji kodu lecz generyczne metakomponenty oraz wiedzę jak z nich skorzystać.

W tej pracy przedstawiono główne założenia i cele podjętego zamierzenia, prace badawcze wchodzące w jego skład oraz wybrane narzędzie – język XVCL. Szczegółowe rozwiązania znaleźć można w cytowanych opracowaniach (Bębenek i in., 2006; Grudzień i in., 2007; Kowalski i in., 2007).

2. Programowanie generatywne i język XVCL

Programowanie generatywne (*generative programming*) jest sposobem tworzenia oprogramowania, wykorzystującym automatyczne tworzenie kodu na podstawie generycznych metakomponentów (klas, prototypów, wzorców, aspektów itp.). Sterowane deklaratywnie generatory z dostarczonego zbioru uniwersalnych metakompo-

nentów tworzą specjalizowane komponenty. Taki sposób budowy oprogramowania może znacząco podnosić produktywność, a przede wszystkim zapewnia znaczne powiększenie stopnia ponownego użycia kodu (*reuse*) i – dzięki oddzieleniu generycznych metakomponentów od specyfikacji sterujących ich specjalizacją – poprawia kontrolę nad tworzeniem kodu. W większości przypadków technika ta powoduje także znaczące zmniejszenie rozmiaru kodu źródłowego.

Choć w pierwotnej intencji programowanie generatywne miało służyć do tworzenia programów w typowych językach programowania, technika ta może być z powodzeniem wykorzystana do generowania dowolnych artefaktów mających reprezentację tekstową, a więc nie tylko programów, ale także różnorodnych specyfikacji, dokumentacji, stron WWW itp.

W badaniach, które tu opisano, wykorzystywany jest język XVCL (*XML-based Variant Configuration Language*, patrz Wong i in., 2001, Soe i in., 2002) – dialekt XML realizujący koncepcję tzw. ram Bassetta (*Bassett frames*, patrz Bassett, 2000).

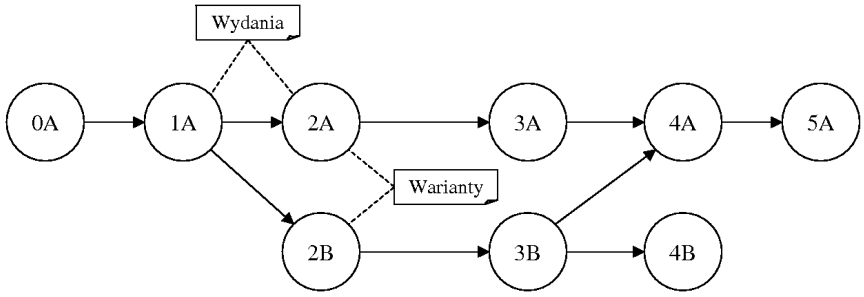
W XVCL zapis mającego wiele wariantów dokumentu składa się ze zbioru plików, zawierających tzw. ramki (*x-frames*). Każda ramka mieści fragment dokumentu, opatrzony znakowaniem XVCL, opisującym zmienność. W ramkach można umieszczać odwołania do innych ramek, przez co ramki tworzą graf skierowany zwany *x-framework*.

Ramki niższych poziomów (adaptowane) zawierają zapisy bardziej generyczne, zaś ramki wyższych poziomów (adaptujące) określają sposób adaptacji swych ramek podrzędnych do bardziej specyficznych warunków. W ramce adaptowanej można określić fragmenty, które w czasie jej przetwarzania mogą być uzupełnione lub zastąpione przez fragmenty zdefiniowane w ramce nadrzędnej (adaptującej). Procesor XVCL przetwarza ramki rekurencyjnie, poczynając od ramki najwyższego poziomu, tzw. ramki specyfikacyjnej. W miejsce każdego napotkanego odwołania do ramki podrzędnej podstawia zaadaptowaną zawartość tej ramki i w ten sposób tworzy kod wynikowy.

Do sterowania adaptacją ramek można używać zmiennych jedno- i wielowartościowych. W tekście adaptowanej ramki można odwoływać się do tych zmiennych i do wyrażeń na nich opartych. Zmienne mogą także sterować działaniem instrukcji XVCL, np. przetwarzaniem warunkowym lub iteracjami.

Użycie XVCL do generowania wersji specjalizowanych komponentów wygląda zatem zwykle tak.

- Buduje się zestaw ramek najniższego poziomu, odpowiadających generycznym metakomponentom, a więc zawierających kod możliwie uniwersalny, silnie sparametryzowany za pomocą instrukcji i zmiennych XVCL.
- Tworzy się wiele ramek najwyższego poziomu (specyfikacyjnych), odpowiadających poszczególnym wersjom (wydanie-wariant), w których specyfikuje się zmienne sterujące adaptacją.
- Tworzy się też jedną lub kilka warstw pośrednich, wspomagających zarządzanie konfiguracją przez wyliczanie pomocniczych zmiennych, specyfikowanie odpowiednich fragmentów do wstawienia itp.



Rysunek 1. Wydania i warianty systemu

- Procesor XVCL wywołuje się, podając mu konkretną ramkę specyfikacyjną, co powoduje wygenerowanie kodu konkretnej wersji komponentu lub zbioru komponentów.

3. Koncepcja zarządzania ewolucją SI z użyciem XVCL

By za pomocą języka XVCL zarządzać wersjami systemu informacyjnego, trzeba wszystkie elementy projektu przedstawić w postaci tekstowej, którą można opatrzyć znakowaniem XVCL. Elementy te to m.in.: opisy wymagań i dokumentacja poszczególnych wersji, sformalizowane modele systemu (np. modele klas UML), struktury baz danych, metadane dla generycznych struktur danych, kod warstwy odwzorowania relacyjno-obiektowego (ORM – *Object-Relational Mapping*), kod wyższych warstw aplikacji (np. komponentów encyjnych i sesyjnych EJB, JSP itd.), specyfikacje i implementacje usług sieciowych (*Web Services*), specyfikacje (np. schematy XML) dokumentów związanych z systemem (np. służących do importu/eksportu danych czy stanowiących zawartość komunikatów SOAP).

3.1. Zarządzanie wersjami

By zapanować nad wielością wersji, trzeba stworzyć metodę formalnego zapisu ewolucji projektu systemu: jego kolejno tworzonych wydań i równolegle współistniejących wariantów (patrz rys. 1). Taki zapis powinien umożliwiać m.in.: (a) przedstawienie kolejnych wydań i równolegle istniejących wariantów systemu; (b) generowanie każdej z wersji systemu lub jego części (także wersji nieaktualnych) oraz dokumentacji odpowiedniej dla danej wersji; (c) łatwe śledzenie ewolucji projektu, w powiązaniu z ewolucją wymagań; (d) ponowne wykorzystanie w przyszłych wersjach wcześniej zaprojektowanych właściwości systemu; (e) tworzenie nowej wersji na podstawie więcej niż jednej z wersji poprzednio istniejących; (f) koordynację zmian wszystkich elementów projektu i warstw systemu; (g) dokumentowanie przyczyn i charakteru wszystkich wprowadzanych zmian.

Utworzenie danej wersji systemu może być spowodowane wieloma niezależnymi czynnikami działającymi równocześnie. Do zarządzania wersjami wybrano więc dość zaawansowany model: wersjowanie oparte na atrybutach (patrz Gergic, 2003).

Każda wersja jest opisana zbiorem par atrybut-wartość, przy czym dla każdego atrybutu zdefiniowane są taksonomie, które wyznaczają związki między dopuszczalnymi wartościami atrybutu, np. relacje następstwa lub generalizacji. Dzięki istnieniu taksonomii można wyznaczyć właściwą wersję nie tylko podając konkretne wartości poszczególnych atrybutów, ale także wykorzystując zależności między wartościami, np. wybranie rozszerzonej funkcjonalności danego podsystemu powinno automatycznie pociągać za sobą wybranie także jego funkcjonalności podstawowej.

Użytkownik, definiując nową wersję lub wyszukując odpowiednią z wersji już istniejących, określa zestaw interesujących go par atrybut-wartość. Odpowiednie narzędzie powinno – wykorzystując wyżej wspomniane taksonomie – wspomóc go w określeniu zestawu parametrów jednoznacznie wyznaczającego wersję oraz na tej podstawie wypracować nową ramkę specyfikacyjną (najwyższego poziomu) XVCL, deklarującą nową wersję za pomocą zmiennych XVCL, lub znaleźć właściwą ramkę dla wersji szukanej.

Należy zauważyć, że ten model wersjowania nie wymusza utrzymywania zależności typu rodzic-dziecko między wersjami. Ponieważ jednak dla śledzenia ewolucji systemu istotna jest znajomość tego typu zależności, należy przyjąć, że wśród wykorzystywanych atrybutów powinny być i takie, który określają wydanie systemu, z taksonomiami definiującymi relacje następstwa.

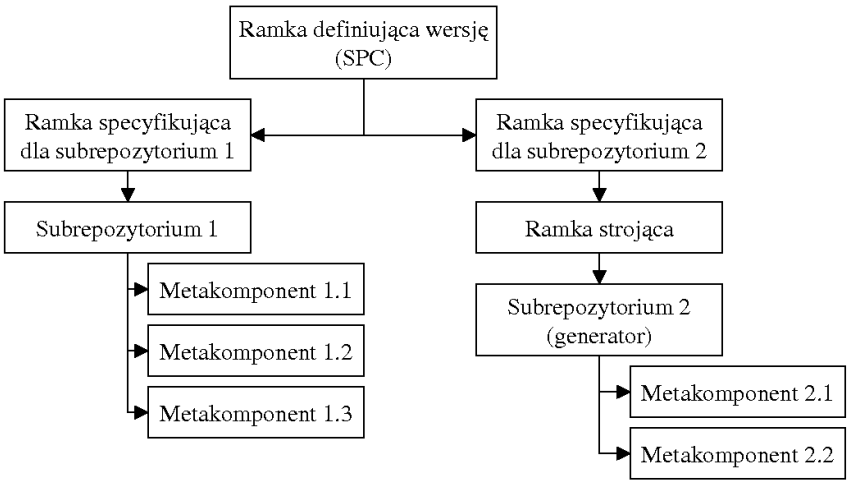
3.2. Repozytorium projektu

Zapisane w postaci zbioru ramek XVCL (czyli *x-framework*) specyfikacje systemu tworzą repozytorium projektu, składające się z wielu subrepozytoriów, odpowiadających poszczególnym warstwom systemu oraz częściom jego dokumentacji (specyfikacji wymagań, diagramom UML itp.).

Każde z tych subrepozytoriów zawiera definicje poszczególnych metakomponentów – każdą w osobnym pliku XVCL. Metakomponenty zawierają szkielety (ewentualnie częściowo wypełnione „stanem początkowym" projektu) poszczególnych składników systemu lub jego dokumentacji. W takim pliku-ramce za pomocą znakowania XVCL oznacza się wszystkie miejsca, które mogą ulegać modyfikacji, np. punkty w których można dodać nowe części definicji. Jeśli w czasie rozwoju systemu okaże się, że trzeba dokonać modyfikacji w miejscu wcześniej nie przewidzianym, to dodaje się odpowiednie znakowanie XVCL. Dotychczas istniejącej treści ani znakowania XVCL nie zmienia się, by zachować możliwość generowania wcześniejszych wersji systemu. Dzięki tak zbudowanym subrepozytoriom cała informacja o możliwych sposobach dostosowania każdego z metakomponentów znajduje się razem, co ułatwia analizowanie wpływu zmian.

Każde subrepozytorium ma swoją ramkę nadrzędną, sterującą adaptacją ramek jego metakomponentów[3]. W czasie generowania systemu główna ramka specyfikacyjna (SPC), definiująca konkretną wersję, adaptuje pomocniczą ramkę specyfikacyjną, w której dokonuje się „tłumaczenia" specyfikacji wersji na „język" właściwy dla danego subrepozytorium, np. wyliczane są wartości pomocniczych zmiennych,

[3] W przypadku subrepozytoriów służących do generacji kodu programów, taką ramkę można nazwać generatorem.



Rysunek 2. Struktura repozytorium projektu

sterujących adaptacją kodu w danym subrepozytorium. Po drodze może także znaleźć się ramka określająca parametry strojące dla użytego narzędzia informatycznego (patrz rys. 2). Ponieważ adaptacją całego systemu steruje ta sama ramka SPC, wersje wygenerowanych fragmentów są zawsze zsynchronizowane.

3.3. Zarządzanie dokumentacją wersji

Gdy podejmuje się decyzje dotyczące modyfikacji systemu informacyjnego, z reguły analizować trzeba jego dotychczasowy kształt oraz przyczyny tego, że jest on taki a nie inny. Oznacza to, że powinna istnieć łatwa metoda odnalezienia przyczyn wcześniejszych decyzji projektowych i istnienia określonej postaci kodu.

W projektach prowadzonych według sformalizowanych metodyk na ogół dość łatwo można powiązać pierwotny kształt systemu z konkretnymi elementami specyfikacji wymagań. Z czasem jednak, gdy system ewoluuje, coraz trudniej jest znaleźć w dokumentacji uzasadnienia wprowadzania poszczególnych zmian. Sprawa komplikuje się jeszcze bardziej, gdy system istnieje w wielu równoległych wariantach. Dlatego wprowadzając zmianę powinno się nie tylko tak zmodyfikować projekt samego systemu, by spełniał on zmienione wymagania, ale także jednocześnie dostosowywać dokumentację.

Za pomocą XVCL można ten problem rozwiązać tak: w ramce specyfikacyjnej „wywołującej" daną wersję powinien się znaleźć opis przyczyn jej zaistnienia oraz syntetyczny opis jej skutków. Dokumentacja techniczna i użytkowa systemu powinna być wersjowana za pomocą XVCL, tak jak inne specyfikacje systemu. W tej dokumentacji powinny znajdować się oznakowane za pomocą XVCL szczegółowe opisy poszczególnych wydań i wariantów oraz zmian/różnic wymagań, które spowodowały powstanie danej wersji; odpowiednio wygenerowane opisy umieszczane byłyby w wynikowej dokumentacji w zależności od generowanej wersji systemu.

Zmiany w dokumentacji mogą zatem być automatycznie synchronizowane ze zmianami oprogramowania. Takie podejście ma ważne zalety: łatwo można prześledzić, które zmiany w dokumentacji są wywołane którymi zmianami wymagań i związane są z którymi zmianami w systemie; można w każdej chwili wygenerować nie tylko specjalizowaną dokumentację do każdego z aktualnych wariantów systemu, ale także do każdej wcześniejszej wersji, z dowolnego wydania; można też wygenerować zestawienia wersji, z opisem przyczyn ich zaistnienia. Należy zauważyć, że możliwość wersjowania dokumentacji, zsynchronizowanego z rozwojem systemu, jest niezbędna do utrzymania kontroli nad ewolucją systemu w dłuższym okresie.

3.4. Zarządzanie ewolucją modeli UML

Ważną częścią specyfikacji większości systemów informacyjnych są modele graficzne. Współcześnie najczęściej używa się modeli zapisanych w UML. Modele te mają reprezentacje tekstowe, które można poddać wersjowaniu za pomocą XVCL.

W ramach tego projektu skupiono się na diagramach klas UML. Podjęte dotąd próby zarządzania wersjami diagramów UML za pomocą XVCL opisano w części 4. Ponieważ dotychczasowy kierunek działania wydaje się być obciążony znacznymi trudnościami, dalsze prace koncentrują się na stworzeniu dialektu XML, specyfikującego diagramy klas UML i dostosowanego do użycia z XVCL, oraz narzędzia wizualizującego takie modele za pomocą języka SVG tak, by można było bezpośrednio (np. w postaci animacji) śledzić różnice między wersjami. Planowane jest także stworzenie narzędzi do konwersji takich modeli z/na język XMI.

3.5. Zarządzanie ewolucją schematów relacyjnych

Kluczowym elementem projektu systemu informacyjnego jest specyfikacja struktury bazy danych, od niej bowiem silnie zależne są szczegóły wyższych warstw systemu: ORM, warstw realizujących logikę biznesową i prezentację, a także reprezentacji danych używanych do wymiany informacji, np. za pomocą usług sieciowych. Choć prawidłowa koncepcja i staranne wykonanie poszczególnych warstw może osłabiać propagację niewielkich zmian (np. drobne zmiany w strukturze relacyjnej mogą wpływać na ORM, ale już nie na model obiektowy, który ów ORM udostępnia wyższym warstwom), jednak w przypadku dużych zmian czy znaczących rozszerzeń funkcjonalnych wpływ modyfikacji struktur danych obejmuje większość warstw systemu. Jednocześnie od jakości projektu struktury, a w szczególności od jej dopasowania do możliwości DBMS, silnie zależy wydajność systemu.

Wersja struktury danych wyznaczana jest przez przynajmniej trzy niezależne czynniki: wydanie wynikające z postępu projektu, konkretne wdrożenie z jego specyficznymi wymaganiami oraz rodzaj i wersję użytego DBMS.

Nie mamy powszechnie uznanego sposobu formalnego modelowania struktur relacyjnych ze wszystkimi szczegółami, które mogą być potrzebne by optymalnie wykorzystać możliwości konkretnego DBMS. Jako sposób opisu schematu przyjęto więc zapis w postaci kodu SQL DDL (*Data Definition Language*).

Ponieważ dialekty języka SQL różnią się znacząco, co odzwierciedla zróżnicowanie istotnych możliwości poszczególnych DBMS, bezpośrednie użycie SQL DDL do stworzenia opisu wersji i wariantów systemu nie jest dogodne. Zaproponowano więc (Kowalski, 2007) specjalny dialekt XML, bazujący na SQL DDL, ale wyróżniający cechy wspólne dla różnych DBMS. Notacja ta umożliwia jednolity zapis i łatwe porównywanie tych cech schematu, które są wspólne dla różnych DBMS (np. tabel i kolumn, indeksów, więzów integralności, uprawnień), ale zachowuje konstrukcje specyficzne (np. kod procedur i funkcji składowanych w języku właściwym dla danego DBMS).

Specyfikacje poszczególnych obiektów bazy danych (tabel, perspektyw itd.) zapisywane są w owym dialekcie, a dopiero ten zapis jest opatrywany znakowaniem XVCL, sterującym wersjowaniem. Procesor XVCL generuje specyfikację wynikowej struktury także tym dialekcie, a z niego tworzony jest przez proste przekształcenie właściwy kod SQL DDL.

3.6. Zarządzanie ewolucją aplikacji

We współczesnych systemach informacyjnych aplikacje zwykle składają się z wielu wzajemnie zależnych warstw. Obiektowy kod aplikacji z relacyjną bazą danych łączy zwykle tzw. warstwa odwzorowania relacyjno-obiektowego (ORM). Wersja tej warstwy wyznaczana jest przez przynajmniej cztery niezależne czynniki: wydanie wynikające z postępu projektu, konkretne wdrożenie z jego specyficznymi wymaganiami, rodzaj i wersję użytego DBMS oraz rodzaj i wersję narzędzia ORM.

Wyższe warstwy aplikacji nie zależą bezpośrednio od struktur danych, ale są od nich zależne pośrednio – przez zależność od ORM. Wersja każdej z warstw aplikacji zależy także od wydania systemu, wymagań związanych z konkretnym wdrożeniem oraz od rodzaju i wersji użytych środków informatycznych (np. wersji EJB).

W omawianym tu podejściu aplikacje mają być generowane automatycznie na podstawie szablonów (metakomponentów) oraz specyfikacji wersji. Warto zauważyć (patrz Bassett, 2000), że użycie ram Bassetta do generowania aplikacji ma ważną przewagę nad użyciem tradycyjnych generatorów kodu. Otóż wynik działania generatora aplikacji wymaga często pewnych drobnych dostosowań (*customizations*). Dla typowego generatora odbywa się to z reguły przez dokonywanie modyfikacji w wygenerowanym kodzie. Ponowne generowanie powoduje zamazanie owych poprawek. Stosując XVCL rozwiązuje się ten problem inaczej: należy adaptowany metakomponent opatrzyć takim znakowaniem XVCL, by ewentualne dostosowania wynikowego kodu mogły być także realizowane jako wynik adaptacji, tj. sterowane z ramek nadrzędnych. Gdy więc jest potrzebne dostosowanie kodu wynikowego, odpowiednie informacje umieszcza się w ramce adaptującej. Przy tym podejściu nigdy nie ma potrzeby ingerowania w kod wynikowy, nie ma zatem także niebezpieczeństwa zamazania wcześniej wprowadzonych modyfikacji.

3.7. Zarządzanie ewolucją schematów XML

Do wymiany danych między systemami informacyjnymi stosuje się często dokumenty XML, o strukturze zdefiniowanej za pomocą standardu XML Schema. Do-

kumenty XML mogą także być używane do przechowywania danych, zamiast lub obok bazy danych, albo nawet wewnątrz niej (np. do przechowywania informacji semistrukturalnych).

Struktura takich dokumentów XML oczywiście zmienia się wraz z rozwojem systemu, a jeśli system istnieje w wielu wariantach, to także dokumenty XML mogą mieć wiele wariantów. W szczególności struktura dokumentów służących do wymiany informacji jest silnie zależna od wersji struktur danych systemu. Ewolucja struktury dokumentów XML odzwierciedla się w wersjowaniu ich schematów XML Schema i może być zarządzana za pomocą XVCL.

3.8. Narzędzia

W ramach opisywanego tu projektu badawczego budujemy pewne prototypowe narzędzia, umożliwiające sprawdzenie funkcjonalności i przydatności zaproponowanych rozwiązań. Kluczową cechą takiego prototypu nie jest wydajność ani przyjazność dla użytkownika, lecz elastyczność: łatwość modyfikacji i dostosowania do udoskonalanych koncepcji. Ważna jest także prostota kodu, by nowy uczestnik prac mógł łatwo włączyć się do działań nad każdym z obszarów projektu. Dlatego podstawowe operacje na repozytorium wykonywane powinny być w językach skryptowych: XVCL oraz XQuery (do takich przekształceń specyfikacji, których nie da się zrobić w XVCL).

W ramach prac nad poszczególnymi częściami projektu podejmowane są także próby stworzenia przyjaźniejszych dla użytkownika narzędzi (patrz część 4), ułatwiających zarządzanie wersjami, porównywanie wyników itp. Takie narzędzia tworzone są w środowisku Eclipse, z wykorzystaniem wtyczki XVCL Workbench (patrz `http://xvcl.comp.nus.edu.sg/`).

4. Dotychczasowe badania i dalsze plany

Dotychczas przeprowadzono próby użycia XVCL do zarządzania ewolucją kilku obszarów projektu systemu informacyjnego. Testowano też różne sposoby zapisu wersji w języku XVCL, w kolejnych pracach udoskonalając stosowane podejście.

Badano możliwość użycia XVCL do zarządzania wersjami diagramów UML (patrz Bębenek i in., 2006). Diagramy tworzono za pomocą programu Rational Rose, zapisywano w języku XMI (*XML Metadata Interchange*) i tę postać opatrywano znakowaniem XVCL. Do wspomagania zarządzania wariantami użyto także narzędzia Rational Rose. Choć pierwsze wyniki były zachęcające, w dalszych pracach pojawiły się trudności przesądzające o zmianie kierunku działań. Otóż okazało się, że kod XMI tworzony przez narzędzie jest bardzo nieporządny, np. przy kolejnych zapisach modelu nie są zachowywane identyfikatory obiektów; zarządzanie tym kodem dla większych projektów byłoby bardzo trudne. Co więcej, narzędzie Rational Rose przestało być rozwijane przez producenta, zatem kontynuowanie prac z jego wykorzystaniem stało się bezcelowe.

Dla struktur baz danych zaproponowano i zbadano (patrz Kowalski i in., 2007) sposób zarządzania ewolucją z użyciem repozytorium o budowie opisanej w 3.2.

Opracowano dialekt XML do opisu SQL DDL, uniezależniający ten opis od platformy DBMS. Stworzono pomocnicze narzędzia informatyczne: parser SQL DDL dla platformy Oracle, umożliwiający przekształcenie skryptów SQL do naszego dialektu, oraz narzędzie do porównywania schematów struktur danych zapisanych w tym dialekcie. Narzędzia te zintegrowano ze środowiskiem Eclipse i wtyczką XVCL Workbench.

Obecnie zaawansowane są prace nad zarządzaniem wersjami warstwy ORM (patrz Grudzień i in., 2007). Zmiany wersji ORM są oczywiście zsynchronizowane ze zmianami struktur danych, zarządzanymi w sposób podobny do opisanego wyżej.

Trwają także prace nad dialektem XML opisującym wybrane diagramy UML (przede wszystkim diagram klas) i narzędziem do ich wizualizacji. W najbliższym czasie planowane jest rozpoczęcie prac nad zarządzaniem wersjami schematów XML oraz nad opracowaniem ujednoliconego repozytorium dla wszystkich części specyfikacji systemu i zestawu narzędzi do obsługi tego repozytorium.

5. Podsumowanie

Użycie przedstawionego w tej pracy podejścia do zarządzania ewolucją systemów informacyjnych może pozwolić na:

- stosunkowo łatwe zarządzanie systemami mającymi wiele jednoczesnych wariantów, uzależnionych od wielu niezależnych czynników (np. od specyfiki użytkownika oraz od rodzaju i wersji użytego oprogramowania);
- uniknięcie „eksplozji" liczby nieznacznie różniących się plików źródłowych dla poszczególnych wariantów;
- rejestrację i możliwość łatwego śledzenia przyczyn wprowadzania poszczególnych wersji;
- tworzenie nowych wersji z więcej niż jednej wersji wcześniejszej;
- ponowne użycie (*reuse*) raz wprowadzonych rozwiązań;
- generowanie w każdej chwili każdej z wersji systemu;
- tworzenie nowych wariantów także dla starszych wydań systemu;
- generację dokumentacji ściśle dostosowanej do danej wersji systemu;
- tworzenie kodu dokładnie dostosowanego do oczekiwań użytkownika, bez potrzeby dokonywania modyfikacji w kodzie po jego wygenerowaniu.

Prezentowane tu podejście różni się od typowego zarządzania wersjami za pomocą ogólnie znanych narzędzi, takich jak CVS czy Subversion. W repozytorium projektu nie przechowuje się bowiem gotowych wersji kodu, lecz generyczne metakomponenty oraz wiedzę na temat tego, jak z owych bazowych metakomponentów uzyskać poszczególne wersje systemu. Reprezentacją danej wersji nie jest więc jej kompletny zapis, ani zapis różnic w stosunku do wersji poprzedzającej, lecz zapis różnic w stosunku do bazowego zbioru generycznych metakomponentów.

Gdy w grę wchodzi zarządzanie ewolucją systemu mającego wiele wariantów, użycie typowych narzędzi nie rozwiązuje ważnych problemów: (a) nie ułatwia opanowania „eksplozji" liczby wersji poszczególnych komponentów, związanych z różnicami między wariantami; (b) nie daje łatwej możliwości wykorzystywania w kolejnych wersjach nowych cech systemu wprowadzonych we wcześniej istniejących wariantach. Podejście oparte na programowaniu generatywnym wydaje się zaś te problemy rozwiązywać w sposób zadowalający. Co więcej, powody i sposób powstania każdej z wersji są tu łatwe do prześledzenia.

Nasze podejście do zarządzania wersjami nie wspomaga pracy grupowej nad oprogramowaniem. Nie ma jednak przeciwwskazań, by to podejście połączyć z użyciem narzędzi typu CVS: repozytorium projektu składa się przecież z plików tekstowych, których modyfikacje mogą być zarządzane przez tego typu narzędzie.

W stosunku do innych metod zarządzania ewolucją struktur danych i oprogramowania, nasze podejście ma pewne zalety: (a) pozwala w jednolity sposób zarządzać zmiennością wszystkich składników projektu, związanych ze wszystkimi fazami rozwoju systemu, od analizy do eksploatacji; (b) jest stosunkowo proste koncepcyjnie, wykorzystuje powszechnie znany język XML i dostępne za darmo narzędzia.

Użycie programowania generatywnego i XVCL do zarządzania ewolucją systemu nie jest oczywiście wolne od wad. Wśród nich można wymienić komplikację kodu źródłowego, wynikającą z obecności znakowania XVCL, nietypowy – różny od powszechnie znanego – sposób podejścia do zarządzania wersjami, czy dużą wrażliwość skuteczności rozwiązania na poprawność i staranność zaprojektowania i utrzymania repozytorium.

Przeprowadzone dotąd próby zastosowania XVCL do zarządzania ewolucją systemu informacyjnego mającego wiele wariantów dają obiecujące rezultaty; szczególnie istotna wydaje się możliwość zapisu wielu zróżnicowanych wariantów systemu bez „eksplozji" liczby nieznacznie różniących się plików źródłowych oraz łatwość odnalezienia przyczyn zaistnienia poszczególnych rozwiązań w danej wersji systemu.

Wykonane prace badawcze objęły jedynie niektóre warstwy systemu i składniki jego specyfikacji. Prace rozpoczęte i planowane mają sprawdzić przydatność zaproponowanego podejścia do zarządzania ewolucją wszystkich składników systemu. Jeśli dadzą one pozytywne rezultaty, dodatkowym wynikiem będzie stworzenie prototypowego narzędzia typu CASE, wykorzystującego język XVCL do zarządzania ewolucją systemu informacyjnego.

Literatura

Bassett, P. (2000) The Theory and Practice of Adaptive Components. Proc. 2-nd Int. Symposium on Generative and Component-Based Software Engineering. *Lecture Notes In Computer Science* 2177. Springer-Verlag.

Bębenek, A., Traczyk, T. (2006) Koncepcja użycia języka XVCL do zapisu ewolucji diagramów UML reprezentujących struktury baz danych. *Bazy danych. Struktury, algorytmy, metody.* 1, 13-22. WKiŁ, Warszawa.

Gergic, J. (2003) Towards a Versioning Model for Component-based Software Assembly. *19-th IEEE International Conference on Software Maintenance ICSM'03*, 138-147.

Grudzień, A., Traczyk, T., Jarzabek, S. (2007) Application of Generative Programming to Evolution of Object-Relational Mapping Layer. *Proc. 2-nd AIS SIGSAND European Symposium on Systems Analysis and Design*, Gdańsk, 64-71.

Jarzabek, S., Basset, P., Zhang, H., Zhang, W. (2003) XVCL: XML-based Variant Configuration Language. *Proc. Int. Conf. on Software Engineering ICSE'03*, Portland, 810-811.

Kowalski, J., Traczyk, T. (2007) Zastosowanie języka XVCL do zarządzania ewolucją schematu relacyjnej bazy danych. *Bazy danych. Nowe technologie.* 1, 35-44. WKiŁ, Warszawa.

Lieberherr, K.J. (1995) *Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns*. PWS Publishing Co.

Soe, M.S., Zhang, H., Jarzabek, S. (2002) XVCL: A Tutorial. *Proc. 14-th Int. Conf. on Software Engineering and Knowledge Engineering SEKE'02*, Italy. ACM Press.

Wong T.W., Jarzabek, S., Myat Swe, S., Shen, R., Zhang, H.Y. (2001) XML Implementation of Frame Processor. *Proceedings of ACM Symposium on Software Reusability SSR'01*, Toronto, Canada.

XVCL Web Page `http://xvcl.comp.nus.edu.sg/`

Evolution management of information system by means of generative programming and XVCL language

Modern information systems with databases are complex products, consisting of many layers and varied software. The systems must, on the long-term horizon, fulfill continuously changing requirements. Managing evolution of such systems is particularly difficult if they have many concurrent variants.
This paper describes conception of evolution management of complex multi-variant information systems with databases, by means of generative programming and XVCL language. It shows results of the research done so far and presents plans of further research work.