KOMPLETNE ARCHIWUM NUMERÓW SDJ I SDJ EXTRA Z 2008 ROKU

Software Developer's JOURNAL 08/2009

Software Developer's JOURNAL

new ideas & solutions for professional programmers

Nr 8 (176) Sierpień 2009 cena 26,75 stawka VAT 0% INDEX: 35497X

KOMPLETNE ARCHIWUM NUMERÓW SDJ I SDJ EXTRA Z 2008 ROKU

GRAILS I GROOVY
CZYLI JAVA JESZCZE ŁATWIEJ

GROOVY
NA POHYBEL JAVIE,
Z MIŁOŚCIĄ DO JAVY
WPROWADZENIE DO JĘZYKA

JĘZYK SKRYPTOWY LUA
CHARAKTERYSTYKA I ROLA
JĘZYKA SKRYPTOWEGO
LUA W PROGRAMOWANIU GIER

SQL SERVER 2008
DATA MINING

WS-BPEL
INSTRUMENTACJA
PROCESÓW BIZNESOWYCH

ANALIZA BIZNESOWA
WYMAGANIA
POZYSKIWANIE,
DOKUMENTOWANIE,
KOMUNIKOWANIE

BIBLIOTEKA MIESIĄCA

GRAILS, GROOVY I RUSZTOWANIA
TWORZYMY APLIKACJE DO BLOGOWANIA

ISSN 1734-3917
08
9 771734 391085

Grails i Groovy

Jeśli wiesz o co chodzi z karteczkami

aplikuj teraz!

www.adocean-global.com/praca

AdOcean®
OCEAN MOŻLIWOŚCI

Jesteśmy zespołem młodych ludzi - naszą pasją jest to, co robimy. Nasz zespół został starannie dobrany i jest główną siłą naszych działań. Ciągle poszukujemy nowych rozwiązań w „Oceanie Możliwości". Wiedza, entuzjazm, solidarność i profesjonalizm osób zatrudnionych w firmie pozwala nam świadczyć usługi o najwyższej jakości. O pracę w AdOcean ubiegają się osoby z najwyższymi kwalifikacjami.

AdOcean Sp. z o.o.
ul. Wołoska 7, 02-675 Warszawa
www.adocean-global.com
tel. (+48 22) 874 41 53
fax (+48 22) 874 41 01

NOWA WIZJA BEZPIECZEŃSTWA

Secure Mail Intelligence! for Domino
wielosilnikowy system dedykowany do ochrony serwerów IBM Lotus Domino

- dziewięć pracujących jednocześnie silników antywirusowych najlepszych producentów
- jeden z najskuteczniejszych silników antyspamowych stworzonych przez Commtouch
- kontrola wycieku treści
- tworzenie zaawansowanych polityk bezpieczeństwa
- zaawansowany moduł filtrowania treści wyłuskujący zagnieżdżone obiekty (treść wiadomości, rodzaj i zawartość załączników, zagnieżdżone załączniki, skompresowane pliki)
- binarna analiza załączników
- otwarta architektura umożliwiająca dodawanie własnych filtrów i akcji
- natywna integracja z serwerem Lotus Domino
- obsługa wersji 6.x, 7.x, 8.x

Secure Mail Intelligence! Encryption Key Manager
silne szyfrowanie i obsługa zewnętrznych PKI w infrastrukturze IBM Lotus Domino

- umożliwia szyfrowanie, deszyfrowanie, podpisywanie i weryfikację podpisu z wykorzystaniem zewnętrznych, profesjonalnych kluczy (w tym kluczy kwalifikowanych i bezpiecznych) w środowisku Lotus Notes / Domino
- korzysta z podpisów zgodnych ze standardem x.509 zamiast wewnętrznych standardów Lotus Notes/Domino bez blokowania standardowych możliwości Lotus Notes/Domino w tym zakresie
- szyfruje i podpisuje w locie pocztę wewnętrzną i zewnętrzną oraz dokumenty, w tym załączniki
- znakuje czasem i weryfikuje znaczniki czasu
- weryfikuje podpisy kwalifikowane
- umożliwia szyfrowanie każdego listu i dokumentu dodatkowymi, specjalnymi kluczami: Security Officer i Server, tak aby można było analizować i kontrolować przesyłaną treść przez upoważnioną osobę bądź wyspecjalizowane programy do kontroli treści
- implementacja pojedynczego logowania i uwierzytelnienia za pomocą PKI
- ochrona i dodatkowe szyfrowanie plików Notes ID
- umożliwia ochronę informacji niejawnych, poufnych i tajnych (wykorzystuje dedykowane rozwiązania sprzętowe)
- zapewnia poufność, integralność i autentyczność poczty

APLIKACJE BIZNESOWE

zarządzanie procesami biznesowymi
zarządzanie relacjami z klientami
zarządzanie wiedzą
zarządzanie dokumentami
zarządzanie projektami
zarządzanie kosztami
zarządzanie incydentami
zarządzanie współpracą
bezpieczna komunikacja
zarządzanie bezpieczeństwem
e-commerce

Producentem rozwiązań Secure Mail Intelligence! jest firma M2 NET S.A., która już od 10 lat dostarcza klientom zarówno rozwiązania do zabezpieczania poczty, jak i aplikacje biznesowe.

Produkty M2 NET wykorzystuje ponad 500 000 użytkowników i dziesiątki klientów – od małych przedsiębiorstw do dużych firm z listy Fortune 500 z ponad 27 krajów.

M2 NET

www.bezpiecznapoczta.pl

SPIS TREŚCI

06 Aktualności
Rafał Kocisz

21 Opis CD

BIBLIOTEKA MIESIĄCA

14 Grails, Groovy i rusztowania
Mariusz Róg
Grails jest prostym środowiskiem z ogromnymi możliwościami. Czytając artykuł, dowiesz się, co ukrywa się w jego wnętrzu, jakiego rodzaju rozwiązania są stosowane oraz jaką rolę w tym wszystkim ma język Groovy. Krok po kroku dowiesz się, jak łatwo i przyjemnie można napisać aplikację internetową, która w większości stworzy się sama.

KLUB TECHNICZNY

22 Bazy danych w Adobe AIR
Mateusz Małczak
Adobe AIR i SQLite. Zapoznajemy się z wbudowaną w runtime AIR obsługą bazy danych.

JĘZYKI PROGRAMOWANIA

26 Groovy – na pohybel Javie, z miłością do Javy – wprowadzenie do języka
David de Rosier
Groovy to dynamiczny język dla środowiska Java, w pełni integrujący się z tym językiem, pozwalając na korzystanie z istniejących bibliotek i szkieletów.

38 WS-BPEL – Instrumentacja procesów biznesowych
Piotr Zerynger
Web Services Business Process Execution Language (WS-BPEL) jest językiem programowania, którego celem jest opisanie procesów zachodzących między partnerami biznesowymi. W artykule zostanie przedstawiona przykładowa procedura BPEL, korzystająca z jednego z ogólnodostępnych w Internecie Web serwisów.

46 Język skryptowy Lua – Charakterystyka i rola języka skryptowego Lua w programowaniu gier (i nie tylko)
Paweł Rohleder
Lua jest obok Pythona jednym z najczęściej wykorzystywanych języków skryptowych w programowaniu gier komputerowych. Skrypty często służą implementacji algorytmów logiki gry, sztucznej inteligencji czy interface'u użytkownika, co ułatwia modyfikację i rozbudowę aplikacji bez konieczności zmiany kodu wynikowego. W artykule przyjrzymy się (poprzez liczne przykłady) charakterystyce języka Lua oraz integracji skryptów Lua w aplikacjach takich jak gry komputerowe.

WARSZTATY

52 SQL Server 2008 – Data Mining
Paweł Wilkosz
Każdego dnia do firmowych baz danych napływają niezliczone ilości informacji. Dane składowane w tabelach zazwyczaj przekładają się na statystyki zakupionych towarów lub usług przez klientów. Ale coraz częściej posiadanie wiedzy, ilu konsumentów zostało obsłużonych w danym czasie, nie wystarcza.

Miesięcznik **Software Developer's Journal** (12 numerów w roku) jest wydawany przez Software Press Sp. z o.o. SK

Dyrektor wydawniczy: Anna Adamczyk

Redaktor naczelny: Łukasz Łopuszański *lukasz.lopuszanski@software.com.pl*

Projekt okładki: Agnieszka Marchocka

Skład i łamanie: Monika Grotkowska *monika.grotkowska@software.com.pl*

Dział produkcji i kolportażu: Alina Stebakow *alina.stebakow@software.com.pl*

Wyróżnieni betatesterzy: Roger Zacharczyk

Nakład: 6 000 egz.

Adres korespondencyjny:
Software Press Sp. z o.o. SK,
ul. Bokserska 1, 02-682 Warszawa, Polska
tel. +48 22 427 36 91, fax +48 22 224 24 59
www.sdjournal.org cooperation@software.com.pl

Dział reklamy: *adv@software.com.pl*

Obsługa prenumeraty: EuroPress Polska *software@europress.pl*

Dołączoną do magazynu płytę CD przetestowano programem AntiVirenKit firmy G DATA Software Sp. z o.o.

Redakcja dokłada wszelkich starań, by publikowane w piśmie i na towarzyszących mu nośnikach informacje i programy były poprawne, jednakże nie bierze odpowiedzialności za efekty wykorzystania ich; nie gwarantuje także poprawnego działania programów shareware, freeware i public domain.

NARZĘDZIA

56 Dogadajmy się – O komunikacji dla programistów
Michał Bartyzel, Mariusz Sieraczkiewicz
Im więcej osób pracuje nad projektem, tym bardziej powodzenie przedsięwzięcia zależy od efektywnej komunikacji pomiędzy nimi. Prace nad projektami IT dotyczą programistów, architektów, analityków. Systemy są udoskonalane przez lata, a zaangażowane osoby zmieniają się. W artykule przyglądamy się aspektom komunikacji, szczególnie dotyczących osób pracujących w projektach IT.

58 Analiza biznesowa – Wymagania – pozyskiwanie, dokumentowanie, komunikowanie
Karolina Zmitrowicz
Etap analizy biznesowej jest jednym z kluczowych momentów w projekcie informatycznym. Jakość analizy i uzyskanych za jej pomocą produktów w znacznym stopniu determinuje jakość kolejnych etapów realizacji systemu. Dobrze zaplanowana i wykonana analiza w znacznym stopniu skraca czas implementacji i testowania oraz pozwala uniknąć problemów związanych z brakami w wymaganiach i niespójnością. Artykuł przedstawia podstawowe elementy dobrego procesu analizy.

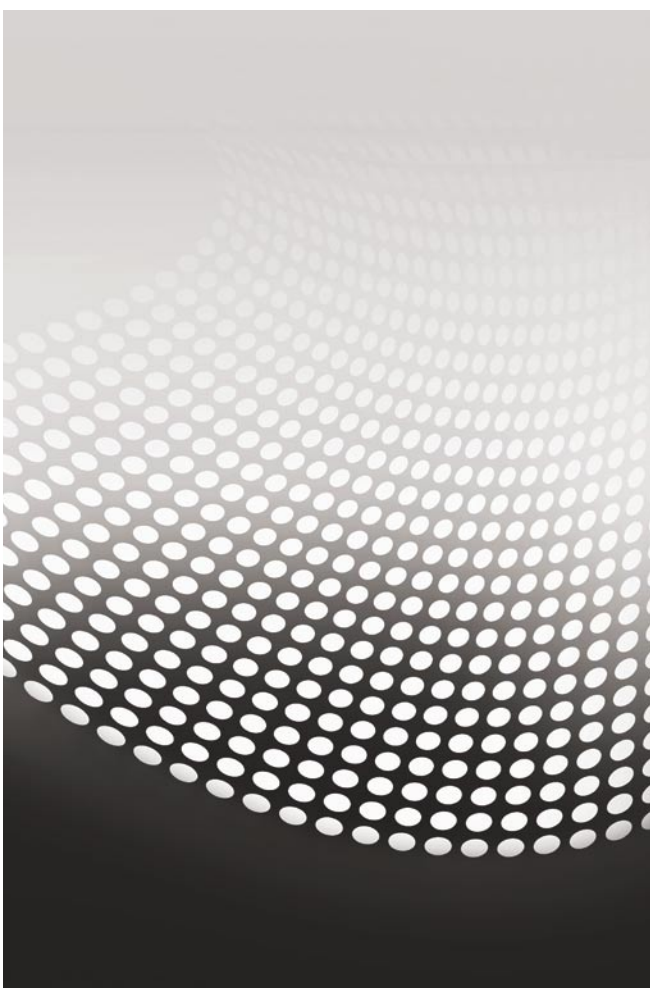
66 Monitorowanie oprogramowania – Lepiej zapobiegać, niż leczyć...
Karolina Zmitrowicz
Coraz częściej instytucje zamawiajce usługi informatyczne posiadają własne departamenty IT i pragną w określonym zakresie uczestniczyć w procesie wytwarzania oprogramowania. Artykuł przedstawia najczęściej spotykane z punktu widzenia klienta problemy związane z taką współpracą oraz propozycje rozwiązania czy uniknięcia owych problemów.

AKADEMIA UML

72 Dziedziczenie – Tworzenie hierarchii klas
Szymon Zioło
W poprzednim odcinku Akademii UML poznaliśmy sytuacje, w których nie należy stosować dziedziczenia. Zobaczmy, kiedy powinniśmy tworzyć wspólną nadklasę dla kilku klas, a kiedy taka hierarchia klas jest zbędna.

PRAWO W IT

76 Prawo autorskie dla programistów – Programista jako twórca dzieła w postaci programu komputerowego
Piotr Przegaliński
W przypadku, gdy programista tworzy program komputerowy, który jest przejawem jego działalności twórczej, ma indywidualny charakter i jest rezultatem jego pracy, wówczas taki program należy traktować jako utwór w rozumieniu Prawa autorskiego. Sam programista powinien być natomiast traktowany jako twórca.



Uszkodzone podczas wysyłki płyty wymienia redakcja.

Wszystkie znaki firmowe zawarte w piśmie są własności odpowiednich firm. Zostały użyte wyłącznie w celach informacyjnych.

Redakcja używa systemu automatycznego składu AUPUS

Osoby zainteresowane współpracą prosimy o kontakt:
cooperation@software.com.pl

Druk: Artdruk *www.artdruk.com*

Wysokość nakładu obejmuje również dodruki. Redakcja nie udziela pomocy technicznej w instalowaniu i użytkowaniu programów zamieszczonych na płycie CD-ROM dostarczonej razem z pismem.

Sprzedaż aktualnych lub archiwalnych numerów pisma po innej cenie niż wydrukowana na okładce – bez zgody wydawcy – jest działaniem na jego szkodę i skutkuje odpowiedzialnością sądową.

Opera Unite: serwer w przeglądarce

Opera Software zaprezentowała technologię, dzięki której każdy użytkownik Opery będzie miał możliwość uruchomienia własnego serwera w przeglądarce. Na początek Opera w ramach technologii Opera Unite udostępniła 6 usług: *File Sharing* (udostępnianie plików z serwera), *Fridge* (zostawianie notatek właścicielowi komputera), *Media Player* (udostępnianie materiałów multimedialnych), *Photo Sharing* pozwalający na przeglądanie materiałów graficznych, *The Lounge* dający możliwość rozmowy pomiędzy zaproszonymi użytkownikami, a usługa *Web Server* pozwoli na proste uruchomienie serwera WWW. Wartym uwagi jest fakt, że Opera nie ma zamiaru ograniczać programisty w tworzeniu nowych usług. W oparciu o te, które już istnieją, możliwe stanie się stworzenie np. własnego silnika blogowego w oparciu o JavaScript, CSS oraz HTML5. Do uruchomienia Opera Unite wymaga konta w usłudze My Opera. Witryny użytkownika będą dostępne w specjalnej subdomenie, np. *[login].operaunite.com*. Dodatkowo użytkownik posiadający więcej komputerów będzie mógł zdefiniować ich dodatkowe adresy dostępowe. Serwer Opery będzie współpracował również z urządzeniami sieciowymi, udostępniającymi protokół UPnP. Dostęp usług, uruchomionych w ramach domowego serwera, będą mieli również właściciele alternatywnych wobec Opery przeglądarek internetowych. Użytkownicy zainteresowani usługą powinni zapoznać się z wprowadzeniem przygotowanym przez deweloperów z Opera Software. Testową wersję przeglądarki Opera z wbudowanym modułem Unite można pobrać ze strony producenta.
http://opera.com/

Bazy danych od Google

Google ogłosiło udostępnienie testowej wersji nowego typu bazy danych, który ma być rewolucyjny w stosunku do tradycyjnych, relacyjnych baz danych. Nowa baza nosi nazwę Fusion Tables i została opracowana w celu uproszczenia wielu operacji, które zwykle są trudne do wykonania przy użyciu baz relacyjnych. Do takich zadań należy np. integracja danych z wielu heterogenicznych źródeł oraz możliwość pracy na dużych zbiorach danych. Fusion Tables wykorzystuje mechanizm *data-spaces* opracowany przez firmę Transformic, którą Google kupiło potem w 2005 roku. Dzięki *data-spaces* ma zostać rozwiązany problem wielu typów i formatów danych. Google Fusion Tables jest dostępne w Labs. Obecnie użytkownik może ładować zbiory danych nie większe, niż 100 MB, a łącznie nie więcej, niż 250 MB. Dane mogą być udostępniane innym użytkownikom.
http://www.google.com/

API dla wyszukiwarki Bing dostępne bez żadnych limitów!

Nowa wyszukiwarka Microsoftu wzbudziła dużą sensację wśród internautów. Po raz pierwszy mamy do czynienia z produktem, który ma szansę zagrozić monopolowi Google'a. Aby uczynić swoją ofertę jeszcze atrakcyjniejszą dla niezależnych deweloperów, gigant Redmond udostępnił pełne API Binga – i to bez żadnych ograniczeń dostępowych. Podobnie jak i sam Bing, tak też API wyszukiwarki jest rozszerzoną wersją istniejącego już interfejsu programistycznego wyszukiwarki Live Search. Nowy interfejs programistyczny jest częścią Projektu Silk Road. Pozwoli on na łatwe rozbudowywanie istniejących witryn lub tworzenie całkowicie nowych aplikacji. Jak możemy przeczytać na stronie Bing Developer Center, praca z nowym API jest bardzo prosta: wystarczy wybrać typ źródła, protokół wyjściowy (obsługiwane są JSON, SOAP i XML) i dostosować całość do naszych potrzeb. Szczegóły wykorzystania poszczególnych źródeł danych opisane są na stronie serwisu MSDN, tu wspomnijmy tylko, że pobrane dane mogą być ograniczone do takich kategorii, jak wyniki przeszukiwania WWW, newsy, obrazki czy odpowiedzi encyklopedii Encarta. Zapytania można zaś wysyłać w formacie zgodnym ze standardem OpenSearch. Co jednak najbardziej ciekawe, to fakt, że Microsoft nie wprowadził żadnych ograniczeń w wykorzystaniu API Binga – aplikacje mogą odpytywać wyszukiwarkę Microsoftu dowolną liczbę razy, pobierać dowolną ilość danych. Jest to wielki krok naprzód w stosunku do Live Search, a także w odniesieniu do innych API oferowanych przez firmy konkurencyjne; warunki nieodpłatnego korzystania z tych ostatnich wiążą się zazwyczaj z poważnymi ograniczeniami. Aby skorzystać z usługi, deweloperzy muszą jedynie wygenerować sobie unikalny identyfikator AppID za pomocą systemu logowania Windows Live ID. Czy to pomoże wyszukiwarce Microsoftu w odbiciu części rynku? Tego nie sposób przewidzieć, ale niewątpliwie za sukces Google'a w wielkim stopniu odpowiada darmowe udostępnianie swoich usług – praktycznie bez żadnych ograniczeń. Najwyraźniej w Redmond postanowiono, przynajmniej w części, przyjąć podobną strategię.
http://webhosting.pl/

Exploit powodem zniknięcia 100000 witryn

Brytyjski dostawca usług hostingowych, firma VAserv, stał się ofiarą ataku hakerów, którzy wykorzystali exploit w popularnym oprogramowaniu wirtualizacyjnym HyperVM, wydanym przez firmę LXLabs. Atakującym udało się skasować praktycznie wszystkie dane na serwerach firmy. Według informacji, które znaleźć można na forum dyskusyjnym poświęconym produktom LXLabs, twórcy oprogramowania wirtualizacyjnego wiedzieli o exploicie od siedemnastu dni i... nic z tą wiedzą nie zrobili. Jak twierdzą użytkownicy forum, firma LXLabs, notorycznie niedostępna dla prasy, traktuje kwestie bezpieczeństwa bardzo lekkomyślnie; potrafi nawet wydawać poprawki, które niczego nie poprawiają. Według Rusa Fostera, dyrektora VAserv, inne firmy, które korzystały z oprogramowania HyperVM, również stały się ofiarami ataku i poważnie na tym ucierpiały. Exploit umożliwił bowiem pełen dostęp do systemu plików serwera i uruchamianie własnych poleceń powłoki, włącznie z rm -rf. Korzystając z tej luki, hakerzy (a może należałoby powiedzieć: wandale) skasowali ponad połowę kont klientów firmy. Teraz na stronie poszkodowanej firmy można jedynie zapoznać się z próbami odzyskania danych prowadzonymi przez administratorów. Niestety, udało im się uratować jedynie część danych. Zawartość niektórych kont została utracona na dobre. Użytkownikom HyperVM, jak również panelu administracyjnego Kloxo/LXAdmin zalecono natychmiastowe wyłączenie tych usług. Jednak dla wielu firm jest to bardzo trudne przedsięwzięcie. HyperVM był często jednym z filarów hostingowej infrastruktury: oprogramowanie to pozwala na zarządzanie bazującymi na hiperwizorach Xen i OpenVZ wirtualnymi serwerami prywatnymi (VPS).
http://webhosting.pl

Bing oficjalnie nową wyszukiwarką Microsoftu

Tak jak planowano, Microsoft podczas konferencji *D: All Things Digital* oficjalnie zaprezentował następcę Live Search: nową wyszukiwarkę o nazwie Bing. Microsoft, mówiąc o Bing, unika słowa *wyszukiwarka*; według giganta z Redmond, Bing to mechanizm wspierający decyzje pozwalający użytkownikom wyjść poza tradycyjne wyszukiwanie. Bing dzięki semantycznemu wyszukiwaniu ma rozwiązać największy problem dzisiejszych wyszukiwarek: 42% początkowych zapytań musi być doprecyzowana. Nowa wyszukiwarka Microsoftu będzie udostępniać dodatkowe wyniki wyszukiwania dopasowane do początkowej frazy tak, że szukane informacje, przynajmniej teoretycznie, powinny pojawiać się na górze bez konieczności zadawania kolejnych pytań wyszukiwarce. Oczywiście, razem z pojawieniem się nowej wyszukiwarki rozpoczęły się na nowo dyskusje i spekulacje na temat przyszłości tego sektora rynku IT. W kilka dni po hucznym uruchomieniu nowej wyszukiwarki Bing i pierwszych badaniach popularności dało się odnieść wrażenie, że następca Live Search odniósł ogromny sukces. Później pojawiły się z kolei głosy, że nowa wyszukiwarka Microsoftu to tylko ogromne rozczarowanie. Atmosferę na rynku wyszukiwarek podgrzała firma StatCounter prowadząca między innymi statystyki popularności przeglądarek. Dane aktualizowane są tam co kilka godzin, więc na bieżąco można było śledzić popularność Binga na tle konkurentów. Kiedy na początku czerwca br. StatCounter opublikował wiadomość, że Bing przegonił Yahoo!, wszyscy, chyba z Microsoftem włącznie, byli zszokowani. Istotnie 4 czerwca udział Bing.com w rynku wyszukiwarek w Stanach Zjednoczonych wynosił ponad 15%, podczas gdy Yahoo! mogło pochwalić się wynikiem jedynie na poziomie 10%. Popularność Binga szokowała zwłaszcza na wykresie – wystrzał był naprawdę imponujący. Dane ogólnoświatowe były bardzo podobne, liczby były niższe, ale przewaga Binga nad Yahoo! równie silna. Ku przerażeniu zwolenników nowej wyszukiwarki, sytuacja pogarszała się jednak z dnia na dzień i już po kilku dniach Bing notował udział w rynku wyszukiwarek w USA na poziomie niższym od Yahoo! Będąc złośliwym, można by powiedzieć, że wszystko wróciło do normy. Bez złośliwości można natomiast przyznać, że kilkudniowy wystrzał popularności spowodowany był wyłącznie chwilowym zainteresowaniem nową wyszukiwarką. Najciekawsze jest jednak to, że popularność Bing.com nadal spada i... jest niższa, niż popularność Microsoft Live Search na miesiąc przed premierą Binga. Zamiast postępu mamy więc regresję. Steve Ballmer zapowiadał, że razem z Bingiem ma zamiar zająć drugie miejsce w rankingach popularności wyszukiwarek, tuż za Google. Dał sobie na realizację tego planu pięć lat. Chyba zbyt piękne byłoby jednak, gdyby ten plan pięcioletni udało się wykonać w tydzień – bez inwestycji, bez przejęć i tak naprawdę bez rewolucyjnych pomysłów. Jeśli ktoś choćby po cichu o tym marzył, to raczej zbyt naiwnie podszedł do całej sprawy. Dość ciekawe spostrzeżenia w sprawie Bing przedstawił dziennik New York Post. Chodzi generalnie o reakcję firmy Google na pojawienie się konkurencyjnego produktu. W pierwszym odruchu przedstawiciele Google wyrażali się o projekcie dość lekceważąco, a do nazwy Bing odnosili się wręcz prześmiewczo. Później jednak stała się rzecz niecodzienna: Sergey Brin, jeden z założycieli Google, powrócił do aktywnych obowiązków bieżącego kierowania firmą, co według wewnętrznych źródeł nie było ostatnio normą. Brin miał rzekomo wykazać bardzo duże zainteresowanie nowymi algorytmami wspomagającymi podejmowanie decyzji, w jakie wyposażony został Bing. Brin powołał specjalny zespół czołowych inżynierów Google, którzy dostali jedno zadanie – opracowanie rewolucyjnych i dających się szybko wdrożyć, zasadniczych nowych funkcjonalności wyszukiwarki. Wygląda więc na to, iż nawet jeśli Bing okaże się niewypałem, być może przysłuży się do zmobilizowania osiadającego ostatnio na laurach Google, w efekcie czego tak czy siak zyskają użytkownicy.

http://www.nypost.com/
http://dobreprogramy.pl/

Microsoft patentuje ocenianie zdjęć

Microsoft chce opatentować ocenianie zdjęć osób, funkcję dostępną od dawna w wielu portalach. W Amerykańskim Biurze Patentowym Microsoft złożył wniosek o patent dla funkcji *Online Personal Appearance Advisor*. Ma to być wynalazek trzech pracowników Microsoft Research. W opisie można przeczytać, że użytkownik wykonuje kilka zdjęć siebie w różnych ubiorach i przy użyciu różnych kosmetyków. Następnie wysyła zdjęcia do odpowiedniego serwisu, gdzie inni użytkownicy mogą głosować na poszczególne fotografie. Po otrzymaniu statystyk użytkownik może wysłać kolejne zdjęcia. Czyżby Microsoft chciał stworzyć własną wersję *fotka.pl*?

http://dobreprogramy.pl/

Bing przetłumaczy strony z wyników wyszukiwania

Usługa wykrywa natywny język użytkownika na podstawie języka przeglądarki. W przypadku gdy strona wyświetlona w wynikach wyszukiwania jest w innym języku, niż język użytkownika, pojawi się odnośnik *Przetłumacz tę stronę*. Domyślnie w dwóch ramkach pojawia się odpowiednio oryginalna oraz przetłumaczona wersja strony. Możliwe jest ustawienie trybu ujednoliconego, w którym oryginalny tekst pojawia się w dymku, po najechaniu danego fragmentu tekstu. Opcja ta działa również w drugą stronę: tłumaczenie tekstu pojawia się w dymku po jego najechaniu.

http://poradnikwebmastera.blox.pl/

Pakiet biurowy od Adobe

Adobe udostępnił finalną wersję usługi Acrobat.com, przy okazji rozbudowując ją o kolejne aplikacje. Równocześnie uruchomiono komercyjną, bardziej rozbudowaną biznesową wersję usługi. Pierwszym składnikiem pakietu jest procesor tekstu Buzzword, który prócz standardowych funkcji (formatowanie tekstu, kopiowanie, wypunktowanie, wyróżnianie, sprawdzanie pisowni) daje możliwość eksportowania dokumentu do różnych formatów. Niedawno do pakietu dołączyły kolejne aplikacje, między innymi narzędzie do tworzenia prezentacji. Aktualnie do dyspozycji mamy ponadto rozbudowany arkusz kalkulacyjny pozwalający na zarządzanie listami zadań, kalendarzem i kontaktami. Koszt podstawowej wersji pakietu wynosi 15 USD. Subskrypcja pozwoli na prowadzenie internetowych spotkań maksymalnie z pięcioma osobami oraz konwersję do 10 dokumentów PDF w miesiącu. Druga, droższa wersja usługi, dostępna w cenie około 40 USD, charakteryzuje się możliwością prowadzenia konferencji z 20 osobami i nieograniczoną ilością konwersji PDF. Oferta dotyczy mieszkańców Ameryki Północnej. Rozmowy pomiędzy 3 osobami pozostają darmowe.

http://www.theregister.com/

GlassFish v3 obsługuje Django

Na łamach bloga projektu pojawił się wpis Viveka Pandeya, szefa obsługi języków skryptowych projektu Glassfish. Można w nim przeczytać: Wersja Preview Glassfisha 3 jest już dostępna, a ja mam przyjemność ogłosić, że obsługuje ona aplikacje zbudowane w Django. Projekt GlassFish-Scripting zapewnia serwerowi aplikacyjnemu odpowiednią implementację kontenera Jythona. Stanowi ona łącznik z modułem OSGi i zapewnia mechanizm automatycznego uruchamiania kodu w kontenerze Jythona w momencie wdrożenia aplikacji napisanej w Pythonie. Dodatkowo kontener Jythona przechwytuje na niskim poziomie zapytania i odpowiedzi HTTP poprzez implementację GrizzlyAdaptera. Pozwala to na większą wydajność środowiska uruchomieniowego i łatwiejszą skalowalność za pomocą framework'u NIO. Kontener Jythona zawiera także WSGI (*Web Services Gateway Interface*), standardowy mechanizm Pythona, za pomocą którego można podłączać go do webowych framework'ów. Dzięki temu w przyszłości obok Django, serwer GlassFish będzie obsługiwał takie frameworki jak Pylons czy TurboGears. Aby uruchomić nowe funkcjonalności, należy skorzystać z narzędzia aktualizacyjnego Glassfisha 3. Autorzy projektu chcą, aby stały się one częścią rdzennej funkcjonalności projektu. Osoby zainteresowane Rubym na pewno ucieszy też wiadomość, że GlassFish 3 w wersji Preview znacznie ulepsza obsługę aplikacji w tym języku. Ich wdrażanie bazuje teraz na Rack, analogicznym do WSGI standardzie komunikacji framework'u Ruby'ego z serwerem webowym. Obecnie Glassfish obsługuje standardowo frameworki Rails, Merb oraz Sinatra.
http://webhosting.pl

Canonical i Hewlett-Packard zaoferują wspólnie serwery

Canonical – komercyjny sponsor projektu Ubuntu Linux – poinformował, że ten system operacyjny otrzyma certyfikaty zgodności z serwerami Hewlett-Packarda, a obie firmy zaoferują wkrótce wspólnie nową, wysoce wydajną konfigurację sprzętu i oprogramowania. Dzięki temu Canonical będzie mógł być instalowany w biznesowych warunkach na siedemnastu modelach serwerów HP ProLiant z linii G6. Według producenta, linia ta to obecnie jeden z najlepszych na rynku produktów – głównym kryterium oceny był tu stosunek wydajności do oszczędności. Nadanie tym maszynom certyfikatu Ubuntu 9.04 Server Edition jest owocem intensywnych testów w laboratoriach Canonical. Dodatkowo Hewlett-Packard zobowiązał się do wydania pakietu obsługi dla Ubuntu, który będzie zawierał sterowniki i aplikacje narzędziowe. Dzięki temu zarządzanie systemem operacyjnym na serwerach amerykańskiego giganta stanie się łatwiejsze. Steve George z Canonical stwierdził, że certyfikowanie serwerów HP to naturalny, kolejny krok w rozwoju Ubuntu – klienci firmy coraz częściej korzystają bowiem z tych maszyn.
http://www.eweek.com

WWDC 2009: Apple rzuca rękawicę Microsoftowi

San Francisco jest niewątpliwie stolicą świata w oczach wszystkich miłośników Apple'a. Podczas trwającej tam Worldwide Developer Conference, firma z Cupertino przedstawiła cały wachlarz nowości, które mają sprawić, że konkurencja – przede wszystkim ta z Redmond – poczuje się pod każdym względem zagrożona. Po pierwsze: Snow Leopard podgryza Windows 7. Microsoft próbuje przedstawić swój Windows 7 jako prawdziwą rewolucję, chwaląc się nowym interfejsem użytkownika i niezwykłą szybkością systemu, tymczasem wiceprezes Apple'a Bertrand Serlet podsumował Siódemkę następująco: *Ten cały Windows 7 to w zasadzie kolejna wersja systemu Windows Vista*. Tymi właśnie słowami Serlet rozpoczął prezentację nowości, jakie Apple zamierza udostępnić klientom w zbliżającej się najnowszej edycji systemu Mac OS X, czyli Snow Leopard. I choć Apple nie ukrywa, że Snow Leopard jest *kolejną wersją Leoparda, którego kochamy i z którego jesteśmy dumni*, to jednocześnie podkreśla, że wprowadzone zmiany nie są tylko kosmetyczne. Gdy Microsoft rozwodzi się nad paskiem zadań, Apple mówi o wykorzystaniu wielordzeniowych 64-bitowych procesorów, wbudowaniu w system obsługi OpenCL (dzięki czemu wszystkie aplikacje będą miały dostęp do mocy obliczeniowej procesora graficznego), natywnej obsłudze serwerów Exchange, rewolucyjnej przeglądarce Safari i udoskonalonych mechanizmach obsługi wideo, dostępnych w ramach nowej wersji QuickTime. Nie tylko jednak funkcjonalnościami Apple licytuje się z Microsoftem. Wojna przenosi się także na grunt cenowy. Okazuje się, że Snow Leopard będzie kosztował posiadaczy Leoparda jedynie... 29 dolarów. Za takie pieniądze Gigant z Redmond na pewno nie będzie chciał sprzedawać Windows 7 posiadaczom Visty. WWDC 2009 okazało się być również świetną okazją na premierę nowej przeglądarki Apple'a. Safari 4 z nowym silnikiem skryptowym Nitro to 64-bitowa aplikacja, która w pełni swoje możliwości rozwinie dopiero w Snow Leopardzie. Ale i tak uruchomienie jej na Windows i w starszych wersjach MacOS X-a poraża szybkością: według testów SunSpidera, przeglądarka Apple'a jest ośmiokrotnie szybsza od Internet Explorera 8 i czterokrotnie szybsza od Firefoksa 3. Microsoft w ostatnich czasach próbował przekonać, że szybkość engine'ów skryptowych nie ma takiego znaczenia, że liczy się czas otwierania stron. Na to Safari 4 ma też swoją odpowiedź: ładuje strony nawet trzykrotnie szybciej niż IE8 i Firefox 3. W dodatku Safari 4 ma być niemal *nie do zdarcia*: Dzięki separacji procesów wtyczek, wszelkie awarie związane np. z Flashem czy Adobe Readerem nie będą miały na nią wpływu. Wystarczy tylko przeładować stronę, by ponownie uruchomić wtyczkę. Nowe Safari to także pełna kompatybilność ze standardami sieciowymi – elementami HTML 5 (m.in. audio i wideo, oraz obsługą aplikacji webowych offline), CSS 3 czy transformacjami SVG. To także test Acid 3 zdany w 100%, poprawki w interfejsie użytkownika i ulepszona prywatność. Safari 4 pobrać można ze strony *www.apple.com/safari/*. Bertrand Serlett przedstawił nowy odtwarzacz multimedialny Apple'a. QuickTime 10 jest kilkukrotnie szybsze od swojego poprzednika, oferuje strumieniowe przesyłanie wideo po HTTP, wykorzystuje kodeki H.264 i AAC, dynamicznie skaluje jakość transmisji do możliwości łącza i potrafi serwować swoje strumienie także przez aktywne zapory sieciowe. Ciekawe, jaka będzie odpowiedź Microsoftu na to wszystko... Jedno jest pewne: druga połowa roku zapowiada się emocjonująco w kontekście rywalizacji dwóch okienkowych gigantów.
http://www.zdnet.co.uk/
http://webhosting.pl/

Microsoft stawia na interoperacyjność między .NET i Javą

Podczas swojego wystąpienia na konferencji JavaOne, Steve Martin, starszy dyrektor platformy produktów dla deweloperów w Microsofcie, potwierdził nieustanne zaangażowanie firmy z Redmond w prace na rzecz interoperacyjności systemów informatycznych budowanych w .NET i Javie. Zapewnił zebranych, że Microsoft przybył na JavaOne w celach pokojowych. Nie ma w tym nic dziwnego – przedstawiciele Redmond od dawna byli zapraszani na JavaOne, zaś rzekoma wrogość między Sunem i Microsoftem jest raczej wymysłem podmiotów nieprzychylnych obydwu firmom, niż rzeczywistość. Odkąd w 2004 roku obie firmy porozumiały się w kwestiach Javy, a Microsoft wypłacił prawie dwa miliardy dolarów za naruszenie własności intelektualnej Suna, stosunki między tymi producentami oprogramowania układają się bardzo dobrze. Jak stwierdził Martin, obie firmy zaczęły wówczas intensywnie ze sobą współpracować na polu technologii. Obsługa Javy w Windows, certyfikaty Windows dla serwerów Suna, otwarcie protokołów oprogramowania Microsoftu, porozumienia prawne i licencje na wykorzystanie elementów Javy i .NET to dowody na to, że od lat Sun i Microsoft zaangażowały się w projekty mające na celu poprawę interoperacyjności. W udzielonym później serwisowi eWeek wywiadzie, Martin stwierdził, że 73% zawodowych programistów polega na wykorzystaniu .NET lub kombinacji .NET i Javy. W tym sensie zarówno Java, jak i .NET wygrały na rynku biznesowym i dlatego jest konieczne, aby zarówno Microsoft, jak i Sun postarały się o to, by interoperacyjność między tymi platformami była czymś rzeczywistym, dostępnym i tak łatwym we wdrożeniu jak to tylko możliwe. Częścią tego zaangażowania ze strony Redmond jest na pewno udostępnienie kodu StockTradera 2.0 do wolnego projektu Stonehenge Fundacji Apache. Demonstruje to zachowanie interoperacyjności pomiędzy wielością software'owych platform, dzięki wykorzystaniu przez nich standardowych protokołów W3C i OASIS. Teraz okazuje się, że Sun również przyłącza się do projektu Stonehenge – przedstawicielka Suna, pani Aisling MacRunnels, podczas wystąpienia dyrektora Microsoftu, poinformowała, że jej firma będzie również dostarczała swój kod do StockTradera. Współpraca między Sunem (czyli de facto już Oracle) a Microsoftem dotyczyć będzie też kwestii *cloud computing*. Martin stwierdził, że sukces w dziedzinie chmur obliczeniowych wiąże się dla większości przedsiębiorstw z możliwością wykorzystania szerokiego spektrum różnorodnych zasobów obliczeniowych, które mogą się między sobą bardzo różnić pod względem wykorzystanych technologii. Kluczem do powodzenia są zatem protokoły interoperacyjności.

http://www.zdnet.co.uk/
http://webhosting.pl

Windows 7 zostanie wydany 22 października

Bill Veghte, zajmujący w Microsofcie stanowisko senior vice president w dziale Windows, zapowiedział, że Windows 7 będzie dostępny dla klientów 22 października br. Rację miał więc dyrektor marketingu Acera, który wcześniej podał datę 23 października. Windows 7 pojawi się prawie dokładnie 8 lat po Windows XP, który został wydany 25 października 2001 roku. Microsoft potwierdził także, że osoby, które kupią komputer z Vistą na krótko przed lub po premierze Windows 7, będą mogły otrzymać za darmo licencję na ten system. Szczegóły jednak nadal nie są znane. Według nieoficjalnych informacji chodzi o okres od 28 czerwca 2009 do 31 stycznia 2010 r. Wersja RTM zostanie udostępniona partnerom Microsoftu w drugiej połowie lipca. Daty pojawienia się w sprzedaży Windows Server 2008 R2 oraz wydania jego wersji RTM mają być zbliżone do dat Windows 7. Oba te systemy powstają bowiem jednocześnie.

http://www.microsoft.com/poland

Microsoft i stan Missisipi zawierają ugodę

Microsoft oraz przedstawiciele stanu Missisipi w USA wspólnie ogłosili zawarcie ugody i tym samym zakończenie ciągnącego się procesu antymonopolowego. Ugoda jest podoba do tych zawieranych z innymi stanami na przestrzeni ostatnich lat. Microsoft zgodził się zapłacić łącznie blisko 100 milionów dolarów wszystkim tym, którzy pomiędzy 1 stycznia 1996 roku a 11 czerwca 2009 dokonali zakupu systemu operacyjnego bądź innej aplikacji produkcji firmy z Redmond. Bony będą mogły zostać wykorzystane do zakupu oprogramowania lub sprzętu. Również instytucje rządowe, które zakupiły produkt w tych latach będą uprawnione do otrzymania bonów pieniężnych. Systemy operacyjne objęte ugodą to MS-DOS oraz Windows 95/98/ME/2000/XP, włączając Windows for Workgroups oraz Windows NT Workstation. Microsoft zapłaci za każdą sprzedaną kopię Windows 95, Windows 98 lub Windows ME dwanaście dolarów pod warunkiem, że nabywcą był mieszkaniec stanu Missisipi i że produkt na terenie tego stanu był używany. Kwota za pakiet Office, w tym jego poszczególne elementy, takie jak Word lub Excel bądź pozostałe systemy Windows, wyniesie pięć dolarów. Microsoft jest zadowolony z osiągnięcia porozumienia z Państwem i Missisippi z naszymi klientami w Mississippi – powiedział Steve Aeschbacher, zajmujący stanowisko radcy generalnego Microsoft. Pieniądze te będą naprawdę pomocne w czasie kryzysu - dodaje Jim Hood z władz stanu.

http://www.cnet.com/

Pierwsza rozwojowa wersja Chrome dla Maca i Linuksa

Google wydało pierwszą wersję deweloperską przeglądarki Chrome przeznaczoną dla użytkowników systemów Linux oraz Mac OS – sam producent jednak odradza póki co korzystanie z niej na co dzień. Chrome dla Maca i Linuksa w dużej mierze to nic innego jak projekt Chromium, tyle że pod inną nazwą. Na blogu twórcy przeglądarki wielkimi literami stanowczo odradzają jednak pobieranie tej wersji - chyba że jest się programistą. To właśnie dla nich skierowana jest ta bardzo wczesna edycja deweloperska. Jest tu sporo racji, codzienne korzystanie z tej wersji Chrome już teraz faktycznie jest raczej utrudnione – nie ma w niej na przykład obsługi wtyczek (nawet Flasha), nie można zmienić ustawień ochrony prywatności, wybrać domyślnej wyszukiwarki czy nawet wydrukować strony. Chromium w rozmaitych testach publikowanych w Internecie wypada całkiem nieźle, jeśli chodzi o wydajność i responsywność. W kwestii funkcjonalności przed programistami Google jeszcze bardzo daleka droga – dzisiaj mamy wersję deweloperską, teraz powstać musi wersja beta, a dopiero potem światło dzienne ujrzy wersja określana jako stabilna.

http://dobreprogramy.pl/

Google: jesteśmy za mali na monopolistę

The Washington Post donosi, że Google przyjmuje nową taktykę w obronie przed nasilającą się presją ze strony władz antymonopolowych USA – chce przekonać amerykański rząd, że jako firma jest zbył mała na monopolistę. Administracja prezydenta George'a Busha dość łagodnie podchodziła do spraw antymonopolowych. Niektórzy uważają, że to właśnie Republikanie uratowali Microsoft przed podzieleniem na dwie firmy na początku tego stulecia. Teraz jednak zmienił się Kongres, zmienił się także prezydent, a z nim przyszedł nowy rząd i nowa szefowa ds. antymonopolowych – Christine Varney. Uważa ona, że nadszedł moment, by dokładnie przyjrzeć się firmie Google, gdyż według niej osiągnęła ona pozycję monopolisty na rynku reklam on-line. Google broni się, przedstawiając się jako malutką firmę, jeśli chodzi o rynek reklamowy. Istotnie, gigant wyszukiwarkowy ma jedynie 2,66% udziały w rynku reklam – uwzględniając telewizję, radio, gazety czy e-maile. Google ponadto argumentuje, że robi wiele dla społeczności, a o wizerunek prawny firmy już wkrótce zadba nie kto inny jak Dana Wagner, była prawniczka ds. antymonopolowych z Departamentu Sprawiedliwości. Jeśli jednak spojrzeć tylko na reklamy w Internecie, faktycznie okazuje się, że Google jest niekwestionowanym liderem. Wątpliwe więc, by rząd USA przychylił się do holistycznej definicji rynku reklamowego nakreślonej przez Google. Raczej władze antymonopolowe skupią się na dominującej pozycji tylko w zakresie reklam on-line, a w takim przypadku do nazwania firmy monopolistą już bardzo niedaleko. Kto wie, jak będzie wyglądać przyszłość firmy z Mountain View – inni gracze, chociażby Microsoft, po zakwalifikowaniu się do grona monopolistów są teraz stale monitorowani, borykają się z niekończącymi się dochodzeniami, procesami i wyrokami. Być może właśnie taki los czeka również Google.

http://dobreprogramy.pl/

Żywność za Internet Explorer 8...

Microsoft uruchomił kampanię Browser for the Better, w ramach której za każde ściągnięcie przeglądarki Internet Explorer 8 przekaże $1,15 organizacji Feeding America. Organizacja ta poprzez sieć banków żywności zaopatruje w artykuły spożywcze 25 milionów Amerykanów. Microsoft chce, aby pomogła ona wyżywić dzieci, które w okresie wakacji są pozbawione darmowych posiłków w szkołach. Ściągnięcia Internet Explorera 8 mają być liczone od 8 czerwca do 8 sierpnia, oczywiście tylko na stronie kampanii. Korporacja chce za nie przekazać do maksymalnie 1 miliona dolarów. Ciekawe, czy ktoś z polskich internautów zamiast kliknąć w pajacyka zechce wspomóc głodujących w USA...

http://www.microsoft.com/poland

Pierwsza stabilna wersja Clojure

4 maja 2009 roku bieżąca wersja Clojure – młodego, lecz dynamicznie rozwijającego się języka programowania dla środowiska Java, została uznana przez jego twórcę, Richa Hickeya, za stabilną i opatrzona numerem 1.0. Clojure rozpowszechniany jest na zasadach zgodnych z Eclipse Public License 1.0. Clojure jest dynamicznym, funkcyjnym językiem programowania kompilowanym do bajtkodu maszyny wirtualnej Javy. Ten pragmatyczny język ogólnego przeznaczenia łączy wysoką efektywność ze zwięzłością, elegancją i prostotą składni oraz interaktywnością właściwą większości języków skryptowych. Doskonale wspiera współbieżność i wielowątkowość. Ściśle integruje się ze środowiskiem Javy, korzystając w pełni i wprost (bez pośrednictwa dodatkowej warstwy translacji) z bogactwa jej zasobów. Dzięki temu rozwiązaniu programy napisane z jego wykorzystaniem osiągają tę samą prędkość działania co programy napisane w Javie. Łączenie pisanych w Clojure i Java części tworzonego projektu jest bezproblemowe. Sam język zawiera stosunkowo niewielką liczbę pierwotnych funkcji (ang. *special forms*), a wszystkie biblioteki standardowe napisane są w Clojure, co daje możliwość ich modyfikacji lub napisania na nowo. Gramatyka i składnia Clojure dziedziczy większość unikalnych cech języka Lisp (np. kod źródłowy jako struktura danych języka, makra), nie będąc jednocześnie obciążona jego balastem historycznym. Wykorzystuje również idee obecne w innych językach programowania, np. ML i Haskell. Zaprojektowana pod kątem jak największej prostoty i użyteczności, pozwala na efektywne skoncentrowanie się na rozwiązywanym problemie i skróceniu czasu tworzenia oprogramowania. Dodatkowym atutem Clojure jest możliwość pracy w trybie interaktywnym oraz wykorzystywania go jako języka skryptowego, co znacznie ułatwia naukę programowania oraz prototypowanie. Dzięki unikalnemu połączeniu wielu funkcjonalności oczekiwanych od nowoczesnego i pragmatycznego języka programowania, społeczność skupiona wokół Clojure bardzo szybko rośnie w siłę. Opracowano już wtyczki dla NetBeans, vim, emacs, powstaje również wtyczka dla Eclipse. Pod koniec maja planowane jest wydanie pierwszego podręcznika pt. *Programming Clojure* autorstwa Stuarta Hallowaya. Póki co, można korzystać z dostępnych w sieci samouczków: Clojure Programming (*http://en.wikibooks.org/wiki/Clojure_Programming*), Learning Clojure (*http://en.wikibooks.org/wiki/Learning_Clojure*), czy Clojure – *Functional Programming for the JVM* autorstwa Marka Volkmanna (*http://jnb.ociweb.com/jnb/jnbMar2009.html#FP*). Dostępne są również nagrania prezentacji Richa Hickeya: *Clojure for Java Programmers* (*http://blip.tv/file/982823*) oraz *Clojure for Lisp Programmers* (*http://blip.tv/file/1313398*). Przystępne wprowadzenie w tajniki programowania w Clojure zawiera również polski serwis clojure.pl (*http://clojure.pl/*).

http://osnews.pl

Bety odstraszają użytkowników?

Wiele wskazuje na to, że być może już wkrótce usługi Google, które przy nazwach posiadają dopisek beta, przemianowane zostaną na wersje finalne. Z tego powodu wiele instytucji wychodzi z założenia, że owe produkty są w fazie testów i odkłada się w czasie ich produkcyjne wdrożenie. Do takich wniosków doszli pracownicy Google podczas odbywającej się w tym tygodniu Google I/O Conference. O ile w przypadku użytkowników domowych oznaczenie to faktycznie przy nazwach wielu usług nie rodzi obaw, o tyle problem pojawia się w przypadku firm, które rozważają inwestycję w wykorzystywanie pewnych usług – bądź co bądź pozostających w fazie testowej. Przykładem takiej aplikacji może być Google Apps, którego co prawda finalna, komercyjna wersja Premier Edition nie posiada oznaczenia Beta, jednak wiele aplikacji wchodzących w jej skład już nie. Najlepiej widocznym tego przykładem mogą być Gmail i inne aplikacje składające się na Google Docs. Pomimo że od ich premiery minęło kilka lat i firma początkowo zapowiadała wydanie wkrótce wersji oznaczonych jako finalne, do tej pory większość pozostaje w fazie bety.

http://arstechnica.com/

Palm Pre: świeża krew w smartfonach

6 czerwca br. miała miejsce być może najważniejsza tegoroczna premiera na rynku smartfonów. Amerykańska sieć Sprint rozpoczęła bowiem sprzedaż modelu Pre, którym firma Palm, bardzo zasłużona w segmencie PDA, zamierza podbić serca klientów. Nowe urządzenie wykorzystuje zupełnie nowy system operacyjny webOS i bywa nazywane zabójcą iPhone'a. Po telefon ustawiły się kolejki, a stosunkowo niewielka liczba egzemplarzy dostarczonych do sklepów (50-60 w każdym) spowodowała, że już po dwóch godzinach sporo zainteresowanych osób nie mogło cieszyć się z zakupu Palma Pre wraz z dwuletnią umową z operatorem. W takiej sytuacji sprzedawcy zaczęli tworzyć listy oczekujących na kolejną dostawę urządzeń. Sprint nie ujawnił jednak, w ilu miejscach telefonów zabrakło. Szacunki co do liczby egzemplarzy, które zostaną sprzedane w czasie bieżącego weekendu, są różne. Oscylują bowiem od ponad 50 do prawie 200 tysięcy sztuk. Klientów do Palma Pre przyciąga przede wszystkim nowy system operacyjny webOS, który jest lepszy od obecnych już na rynku (Symbian, Windows Mobile, iPhone OS, Android, BlackBerry). Autorom oprogramowania Pre przyświecała idea, żeby poszczególne czynności dało się zrobić lepiej i wygodniej. Jako sztandarowy przykład wcielenia jej w życie przywoływana jest możliwość przełączania się między jednocześnie uruchomionymi aplikacjami bez konieczności przechodzenia przez menu startowe. W telefonie jest wreszcie możliwe to, co na zwykłym komputerze jest oczywistością. Użytkownik jest też np. powiadamiany o przychodzącej poczcie za pomocą okna typu pop-up i jeśli chce ją przeczytać, to może po prostu to zrobić od razu. Integracja kontaktów pochodzących z wielu miejsc to następna dobra cecha systemu opracowanego przez Palma. Poczta firmowa, prywatny e-mail czy serwis społecznościowy: każde z tych źródeł jest równie dobre, a nie powoduje powielania tych samych informacji przypisanych do jednej osoby. Na polu sprzętowym Palm Pre nie wyróżnia się jakoś specjalnie na tle konkurencji. Modnie zaokrąglone rogi czarnej obudowy, wysuwana ku dołowi klawiatura QWERTY, ekran o przekątnej 3,1 cala (dla porównania: iPhone ma 3,5 cala) i rozdzielczości 320x480 pikseli (24-bitowy kolor) oraz wbudowana pamięć o pojemności ośmiu gigabajtów. Poniżej ekranu umieszczona jest też powierzchnia czuła na dotyk, która może służyć np. do cofnięcia się o jeden poziom w menu za pomocą prostego ruchu palcem. Jak wykazały testy, telefon pobiera mniej energii otrzymujące e-maile natychmiast (obsługuje bowiem technologię push), niż sprawdzając skrzynkę co jakiś czas, choć jest to raczej wbrew intuicji. Napotkano także na pewne problemy z zarządzaniem energią, ale Palm obiecał je usunąć. Intensywnie używany Pre wytrzymał bez ładowania dwa dni, co jest dość dobrym rezultatem. Jeśli ktoś ma zbędne 70 dolarów, to akumulatory tego smartfona można uzupełniać za pomocą wygodnej ładowarki indukcyjnej; w końcu jeśli urządzenie ma być bezprzewodowe, to niech będzie takie do końca. Oczywiście w urządzeniu nie zabrakło aparatu fotograficznego, który zbudowano na bazie matrycy z trzema milionami punktów. Jego zaletą jest minimalne opóźnienie między momentem naciśnięcia spustu migawki a wykonaniem zdjęcia. Wyposażenie Pre dopełniają moduły Wi-Fi, GPS oraz akcelerometr. 199 dolarów – taką właśnie sumę wykładają chętni na Palma Pre na dzień dobry. Następnie muszą płacić miesięczny abonament nie niższy niż 69,99 dolarów, ale w zamian otrzymują nielimitowaną transmisję danych. Sprint ma wyłączność na sprzedaż Pre do końca 2009 roku i jego najważniejsi konkurenci – Verizon Wireless i AT&T – muszą zaczekać. Co ciekawe, jak wynika z informacji z jednego ze sklepów w San Francisco, mniej więcej połowa klientów kupujących Pre ma już iPhone'a. Jak możemy się z kolei dowiedzieć z serwisu Rapid Repair, który natychmiast rozłożył nowego Palma na czynniki pierwsze, wartość elementów użytych do skonstruowania smartfonu wynosi ok. 170 dolarów – urządzenie jest więc dotowane w niewielkim stopniu albo wcale. Przyszłość Pre rysuje się raczej w dobrych barwach, ale nie można zapomnieć o jego słabościach. Nie wiadomo, jak szybko rozwinie się rynek aplikacji; na razie można wybierać spośród około 20 propozycji. Jeden z tych programów służy do uruchamiania tysięcy aplikacji napisanych dla starszych wersji systemów operacyjnych opracowanych wcześniej przez Palma.

http://www.palm.com/

Microsoft proponuje Windows bez Internet Explorera

Niemożliwe? A jednak – w związku z prowadzonym dochodzeniem antymonopolowym Microsoft rozesłał do producentów sprzętu informację, że planuje rozwiązanie problemu poprzez wydanie specjalnej wersji systemu Windows 7 całkowicie pozbawionej przeglądarki Internet Explorer. Jeśli plany zostaną wcielone w życie, nowa wersja systemu nazwana zostanie Windows 7 "E", i podobnie jak wersja "N" pozbawiona odtwarzacza Windows Media Player, również i ta dostępna będzie jedynie na rynku europejskim. Jakiś czas temu pisaliśmy, że Internet Explorer 8 można wyłączyć (częściowo odinstalować) w Windows 7, ale wersja "E" ma pójść o krok dalej – w ogóle nie będzie zawierać przeglądarki. Najważniejsze jest jednak to, że w przeciwieństwie do wersji "N", tym razem na rynku europejskim nie będzie alternatywy, czyli pełnej wersji z przeglądarką. Integratorzy OEM i sprzedawcy sprzętu będą mogli otrzymać specjalne pakiety instalacyjne, dzięki którym w miarę potrzeby możliwa będzie preinstalacja przeglądarki na nowych komputerach. Sprawa będzie nieco bardziej skomplikowana dla użytkowników, którzy samodzielnie zakupią system – Microsoft planuje dla nich przygotować wersje instalacyjne Internet Explorera na serwerach FTP i dostępnych oddzielnie w sprzedaży płytach CD. Trudno przewidzieć, jak zareaguje na te informacje Komisja Europejska. Jeśli zastanowić się jednak nad tym nieco dłużej, to można dojść do wniosku, że wbrew pozorom może to być spore utrudnienie dla użytkowników, a brak jakiejkolwiek przeglądarki w systemie niekoniecznie może być tym, o co chodzi władzom antymonopolowym Unii Europejskiej...

http://www.cnet.com/

Klapa MSN w Polsce, wkrótce przejęcie przez Agorę

Jak donosi serwis Internet Standard, polska wersja portalu MSN zostanie wkrótce przejęta przez Agorę i wcielona do rodziny serwisów Gazeta.pl. Premiera polskiej wersji serwisu MSN w czerwcu ubiegłego roku nie wywołała burzy, a sam serwis niewiele mógł zaoferować czytelnikowi. Pomimo aspiracji do roli portalu horyzontalnego na równi z innymi tuzami polskiego Internetu, MSN.pl zgromadził jedynie milion użytkowników według marcowej edycji badania Megapanel PBI/Gemius. Po roku funkcjonowania polskiej wersji trudno więc zaliczyć projekt do udanych. W związku z tą sytuacją i brakiem pomysłu na dalszy rozwój portal już wkrótce przejdzie do rąk Agory, która jak dotąd razem z MTV i Sony BMG była głównym dostawcą treści w serwisie Microsoftu. Będzie on dostępny pod nazwą msn.gazeta.pl, a całość pracować będzie w oparciu o technologie Windows Live. Gazeta uzyska także wyłączność na sprzedaż i rozwijanie oferty reklamowej polskich produktów internetowych Microsoftu.

http://www.internetstandard.pl/

Obowiązkowa cenzura Internetu w Chinach

The Wall Street Journal donosi, że rząd w Chinach planuje od 1 lipca wprowadzenie obowiązku dołączania oprogramowania do cenzury Internetu do każdego sprzedawanego komputera. Narzędzie, które ma być preinstalowane na każdym komputerze sprzedawanym w Chinach lub przynajmniej dołączane na płycie nazywać się będzie Green Dam-Youth Escort. Rząd i producent oprogramowania zapewnia, że celem cenzury jest ochrona młodzieży przed nieodpowiednimi treściami – głównie pornografią. Narzędzie będzie działać w połączeniu z centralną bazą zablokowanych witryn, na bieżąco aktualizowaną – jeśli użytkownik zechce wyświetlić witrynę znajdującą się na liście, dostęp będzie zablokowany. Zagraniczni eksperci po zapoznaniu się z oprogramowaniem ujawnili, że narzędzie oprócz cenzury może także wysyłać do producenta informacje osobiste, powodować usterki w funkcjonowaniu komputera lub uczynić go bardziej narażonym na ataki.

Nie wiadomo też, co dokładnie i na jakich warunkach ma być blokowane. Tak czy inaczej, od 1 lipca sprzedawcy sprzętu będą musieli zgłaszać rządowi, ile komputerów z cenzurującym oprogramowaniem sprzedali. W ubiegłym roku w Chinach sprzedano 40 milionów pecetów.

http://dobreprogramy.pl/

Google Squared uruchomione

Zgodnie z zapowiedzią Google uruchomiło nową wersję swojej wyszukiwarki, prezentującą wyniki w postaci tabeli. Google Squared zostało stworzone z myślą o znajdowaniu informacji na temat nie konkretnej rzeczy, ale grupy elementów. Przykładem mogą być np. dystrybucje Linuksa, wersje Windows czy też cyfrowe lustrzanki Canona. Jak widzimy, stosowane jest tutaj wyszukiwanie semantyczne, ponieważ wyszukiwarka rozpoznaje, o jaką kategorię pytamy i odpowiednio dostosowuje kolumny. Tak więc dla Windowsów jest kolumna z ceną, a dla Linuksów kolumna z licencją. W przypadku postaci historycznych znajdziemy kolumny z datą urodzenia i śmierci. Z kolei dla państw są takie kolumny jak stolica, liczba ludności czy religie. Można też dodawać własne kolumny, wpisując je lub wybierając spośród proponowanych. Nowa wyszukiwarka jest jeszcze we wczesnej fazie rozwoju i nietrudno zauważyć, że wyszukiwanie semantyczne sprawia jej problemy. Wyników jest często zaskakująco mało, a dane z poszczególnych kolumn bywają nieprawdziwe. Należy jednak oczekiwać, że jakość wyników wkrótce ulegnie poprawie.

http://dobreprogramy.pl/

Nowy iPhone 3G S, nowe możliwości

Najnowsze dziecko filmy Apple, nowy iPhone 3G S trafił do sprzedaży 19 czerwca br. Litera S w nazwie modelu oznacza szybkość (ang. *speed*), czyli główną cechę przypisywaną temu urządzeniu. Premierę nowego modelu poprzedziły spekulacje w Internecie i precyzyjna kampania firmy Apple. Kontrolowane przecieki i niekończące się plotki podgrzewały atmosferę wokół nowego smartfona. W końcy przyszedł czas na oficjalną premierę podczas World Wide Developers Conference. Jaki jest nowy iPhone? Design najnowszego modelu tylko nieznacznie różni się od wcześniejszego iPhone'a 3G. Natomiast całkowicie zmieniono wnętrze urządzenia. Bezpośrednie porównanie obu modeli wskazuje, że 3GS może być nawet dwukrotnie szybszy od swojego pierwowzoru. Główne cechy nowego iPhone'a to: wbudowany aparat 3MP, nagrywanie filmów w trybie VGA z prędkością 30 klatek na sekundę (filmy można przesyłać bezpośrednio do serwisów wideo lub za pomocą MMS-a), sterowanie głosem, wbudowany kompas, wsparcie dla OpenGL|ES 2.0, wszystkie funkcje dostępne w iPhone 3.0. Nowy model może się pochwalić poprawioną wydajnością baterii. Umożliwia ona 10 godzin oglądania wideo, 30 godzin słuchania muzyki lub 5 godzin rozmów przez 3G. Wstępne ceny modelu to 199 dolarów za wersję 16 GB i 299 dolarów za mocniejsze 32 GB. Premierę w sklepach przewidziano na 19 czerwca. Smartfon trafi w pierwszej kolejności do USA, Kanady, Francji, Niemiec, Włoch, Hiszpanii, Szwajcarii i Wielkiej Brytanii. Później przyjdzie czas na kolejne kraje. Wprowadzenie nowego modelu oznacza obniżkę cen poprzedniego iPhone'a 3G. Za wersje z 8GB trzeba będzie zapłacić 99 dolarów. Programistów, a szczególnie programistów gier, ucieszy zapewne najbardziej fakt, iż nowy smartfon Apple otrzymał układ graficzny wspierający bibliotekę OpenGL ES 2.0. Model G3 oferował zaledwie OpenGL ES w wersji 1.1. Za grafikę w iPhone 3G S odpowiedzialny jest układ PowerVR SGX. Cóż, wygląda na to, że Apple nie żartowało, mówiąc o iPhone jako o potencjalnie najlepszej platformie dla mobilnych gier. OpenGL ES 2.0 jest zupełnie nową, inną niż wspomniana wersja 1.1, wersją API, stąd nowe wyzwania przed programistami – aby wykorzystać moc układu i bibliotek, trzeba będzie napisać zupełnie nowy kod (obie wersje GL nie są ze sobą kompatybilne). Szybszy rendering, programowalny potok graficzny pozwalający uzyskiwać zupełnie nowe efekty (np. zaawansowany postprocessing w oparciu o programowalne shader'y), znacznie bardziej zaawansowane tekstury i dużo, dużo więcej... Litery ES w nazwie biblioteki oznaczają specjalną wersję OpenGL dostosowaną do urządzeń ultramobilnych. Wygląda na to, że przy takich możliwościach pierwotne zastosowanie smartfona może zejść na bardzo daleki plan.

http://www.newsweek.pl
http://www.frazpc.pl

Netbeans 6.7 RC 1

Firma Sun Microsystems udostępniła wersję RC 1 otwartego środowiska programistycznego NetBeans 6.7 Wersja ta zawiera wiele nowości i udogodnień, m. in. została zintegrowana z platformą Kenai umożliwiającą wygodną współpracę wielu zespołów programistów nad projektem, wprowadzono obsługę narzędzia Maven do kontroli buildów, zintegrowano środowisko z serwerem aplikacji GlassFish oraz narzędziem Hudson. Wprowadzono również rozszerzenia w modułach języków Java, PHP, Ruby, Groovy i C/C++, dodano też obsługę JavaScript 1.7.

http://www.netbeans.org/

Flex i Flash: unifikacja środowiska deweloperskiego Adobe

Koniec z Flex Builderem: Adobe Systems zamierza zintegrować swoje narzędzia do projektowania i programowania. Wszystko po to, aby promować świadomość marki, jaką jest Flash, i pozbyć się zamieszania związanego z funkcjonowaniem obok siebie produktów, których zakres wykorzystania nie był dla deweloperów oczywisty. Adobe ogłosiło dlatego wydanie narzędzia do budowy zaawansowanych interfejsów użytkownika Flash Catalyst (znanego wcześniej pod nazwą kodową Thermo). Wydaniu temu towarzyszyć będzie Flash Builder 4 – nowa odsłona bazującego na Eclipse środowiska deweloperskiego Adobe, które znane było niegdyś jako Flex Builder. Mihai Corlan, ewangelista Adobe, tak wyjaśnił przyczyny tej zmiany nazwy: *Kiedy pytam na konferencji ludzi – 'kto z Was jest programistą Flex'a?* – wielu ludzi nie jest pewnych, co odpowiedzieć. Dlaczego tak jest? Bo niektórzy z Was używają framework'a Flex do budowy aplikacji AIR lub Flex, ale nie naszego IDE (Flex Buildera), a niektórzy używają Flex Buildera, ale nie korzystają z framework'a Flex (wykorzystują ActionScript 3 do tworzenia aplikacji we Flashu. Stąd ta niepewność). Ludzie do tej pory myśleli często, że istnieje jakaś fundamentalna różnica pomiędzy Flex'em a Flashem – wyjaśnił Corlan. Jednak aplikacje Flex'a są po prostu kompilowane do pliku SWF i odtwarzane za pomocą Flash Playera. Dzięki nowemu nazewnictwu możliwe będzie także zakreślenie różnicy pomiędzy płatną wersją Flex'a a opensource'owym framework'iem, udostępnianym przez firmę za darmo. Jako że zaś spośród wszystkich produktów firmy najbardziej rozpoznawalną markę ma Flash Player, zdecydowano się połączyć całą rodzinę narzędzi pod nazwą Flash. Ma to pomóc Adobe w walce z kolejną generacją Silverlight'a, której wydanie zapowiadane jest już na lipiec bieżącego roku. A co nowego we Flash Builderze 4? Przede wszystkim widać w nim efekty współpracy pomiędzy Adobe a Zend Technologies. Nowa wersja IDE zawiera wbudowany silnik Zend, który ma ułatwić łączenie flashowych front-end'ów aplikacji z back-end'em tworzonym w PHP. Dostępne są także nowe łączniki dla back-end'ów tworzonych w Javie i uruchamianych na serwerach ColdFusion, a wszystko obsługiwane jest poprzez wygodny interfejs typu *przeciągnij i upuść*. Dodano także mechanizmy, dzięki którym możliwe jest bezpośrednie przenoszenie obiektów pomiędzy Catalystem a Builderem; będą one synchronizowały poprawki wprowadzone w jednym z tych programów z kodem zawartym w drugim. Ma to być możliwe dzięki nowemu formatowi wymiany danych, który firma nazwała FXG. Jedno jest pewne: Adobe nie powiedziało ani słowa na temat możliwej integracji swoich narzędzi z konkurencyjną platformą .NET. Osoby zainteresowane technologiami firmy Adobe mogą pobrać Flash Builder 4 beta ze strony *http://labs.adobe.com/technologies/flashbuilder4/*. Flash Catalyst jest zaś dostępny na stronie *http://labs.adobe.com/technologies/flashcatalyst/*.

http://webhosting.pl

http://www.adobe.com/

USB 3.0 w Linuksie

Kilka miesięcy po tym, jak światło dzienne ujrzała wersja robocza specyfikacji Super Speed USB (USB 3.0), NEC zaprezentował pierwszy gotowy kontroler zgodny z najnowszą wersją specyfikacji. Jak się okazuje, nie trzeba było długo czekać na pierwszy sterownik do obsługi urządzeń wykorzystujących ten standard pod Linuksem. Autorką sterownika jest Sarah Sharp, pracująca w dziale Centrum Technologii Open Source w Intelu, który zaprezentował projekt specyfikacji USB 3.0. Sterownik dołączony został do kolejki sterowników oczekujących na dodanie w jądrze 2.6.31, którego premiera planowana jest we wrześniu. W tym samym czasie na rynku pojawić się mają pierwsze urządzenia wykorzystujące USB 3.0. Wkrótce potem sterownik znajdzie się najprawdopodobniej w najpopularniejszych dystrybucjach Linuksa, między innymi Ubuntu. Linux będzie pierwszym systemem oferującym oficjalne wsparcie dla tego standardu. Osoby zainteresowane obsługą USB 3.0 mogą skompilować ręcznie sterownik, korzystając z odpowiednich repozytoriów.

http://www.techit.pl/

MySpace zwolniło jedną trzecią pracowników

Mimo nie najgorszej kondycji finansowej i wciąż obserwowanego wzrostu, MySpace postanowiło zwolnić 30% swojej załogi. Firma chce rzekomo powrócić do kultury startupowej. Najprawdopodobniej szuka jednak tylko dobrego uzasadnienia do wprowadzenia radykalnych oszczędności. Z pierwszych, nieoficjalnych informacji wynikało, że MySpace może zwolnić nawet połowę z 1,5 tysiąca zatrudnianych osób. Ostatecznie z pracą pożegna się niemal pięćset. Cięcia będą dotyczyć w takim samym stopniu wszystkich amerykańskich oddziałów spółki. Nie jest do końca jasne, czy redukcje obejmą zagraniczne filie. Stosunkowo najbardziej prawdopodobne jest to w Europie. Owen Van Natta, dyrektor generalny MySpace, stwierdził, iż zdaje sobie sprawę z tego, że zaproponowane zmiany są bardzo bolesne, aczkolwiek w jego opinii ruch ten jest nieodzowny po temu, aby odbudować innowacyjne środowisko, które skoncentruje się na użytkowniku i produkcie. Van Natta pracował wcześniej dla Facebook'a. Zastąpił w kwietniu w MySpace Chrisa DeWolfe'a, którego do dobrowolnej rezygnacji namówił nowy supermenedżer do spraw Internetu w News Corp (właściciel MySpace), Jonathan Miller. Tej czystki spodziewano się zresztą już od roku. Z drugiej strony nieoficjalnie wiadomo, że ogłoszone przez Van Nattę cięcia nie będą dotyczyć kadry zarządzającej wysokiego szczebla. Jej celem będzie utrzymanie przychodów, przede wszystkim odnowienie na korzystnych warunkach umowy reklamowej z Google.

http://mashable.com/

DOSBox 0.73, nowy emulator starego DOSa

Po około dwóch latach prac wydano kolejną wersję kultowego już emulatora systemu operacyjnego DOS – DOSBox 0.73, za pomocą którego można uruchomić starsze programy i gry nie działające poprawnie lub nawet w ogóle nie uruchamiające się pod nowymi systemami Windows. Pomimo że zmiana oznaczenia w stosunku do poprzedniej wersji nie wskazywałaby na olbrzymie zmiany, to jednak nowa odsłona programu zawiera całkiem sporą ich listę. Nowościami w tym wydaniu są wsparcie dla systemu operacyjnego Windows Vista oraz szereg poprawek mających na celu poprawę stabilności i szybkości pracy. Jako te najważniejsze wymienić należy wprowadzenie wsparcia dla większej ilości trybów i kart graficznych, nowy rdzeń emulacji OPL, aktualizację modułów dźwiękowych dla systemu Mac OS X, rekompilację rdzenia oraz mnóstwo poprawek zgodności. Ponadto dodano podstawową obsługę sterownika edev klawiatury, łatki do obsługi pamięci (EMS/XMS) oraz zaimplementowano specjalne tryby pracy dla układów graficznych S3, Paradise i Tseng. Szczegółową listę zmian znaleźć można na oficjalnej stronie projektu.

http://www.dosbox.com/

Grails, Groovy i rusztowania

Tworzymy aplikacje do blogowania

Grails jest prostym środowiskiem z ogromnymi możliwościami. Czytając artykuł, dowiesz się, co ukrywa się w jego wnętrzu, jakiego rodzaju rozwiązania są stosowane oraz jaką rolę w tym wszystkim ma język Groovy. Krok po kroku dowiesz się, jak łatwo i przyjemnie można napisać aplikację internetową, która w większości stworzy się sama.

Dowiesz się:
- Czym jest środowisko Grails;
- Do czego służą poszczególne typy klas i plików;
- Jak prosto stworzyć aplikację przy użyciu rusztowania.

Powinieneś wiedzieć:
- Czym są języki dynamiczne;
- Co to jest ORM oraz Hibernate;
- Czym jest wzorzec MVC;
- Jaka jest zasada działania kontrolerów w aplikacjach internetowych;
- Czym jest JSP oraz JSTL.

Poziom trudności

Każdy programista dąży do uproszczenia mechanizmów działających w jego aplikacji. Zasada ta implikuje tworzenie coraz większej liczby nowych środowisk. Bardzo często wymagają one żmudnej konfiguracji oraz ogromnej liczby bibliotek, które i tak nie są potem używane. Często te środowiska różnią się między sobą przyjętą konwencją i nie przedstawiają spójnych mechanizmów. Na szczęście środowisko Grails nie należy do tej grupy, co więcej, bardzo przyjemnie rozwiązuje przedstawione problemy.

Czym jest Grails

Grails jest prostym środowiskiem tworzenia aplikacji internetowych. Jest odpowiedzią na *ciężkie* środowiska, których stosowanie sprowadza się do stwierdzenia: przerost formy nad treścią. Grails pozwala znacznie przyspieszyć tworzenie aplikacji internetowych dzięki koncepcjom zaciągniętym ze środowisk dynamicznych, takich jak Ruby on Rails (dla języka Ruby) czy Django (dla języka PHP). Jest zbudowany w oparciu o środowiska Hibernate i Spring Framework przy wykorzystaniu języka dynamicznego Groovy. Dodatkowo środowisko oparte jest na mechaniźmie wtyczek, dzięki czemu dodatkowe funkcjonalności można dołączać do środowiska wedle potrzeby (podobnie z ich wyłączaniem). Środowisko Grails podobnie jak środowisko Ruby on Rails umożliwia użycie zestawu przydatnych poleceń, dzięki którym możemy stworzyć wiele, nie używając żadnego IDE (ang. *Integrated Development Environment*). Spis najpopularniejszych poleceń widzimy w *ramce*. Polecenia wykonywane są przez narzędzie Gant, które jest odmianą narzędzia Apache Ant. Różnica między tymi narzędziami polega na tym, że Ant używa plików XML do zdefiniowania logiki, zaś Gant używa skryptów napisanych w języku Groovy.

Domyślna instalacja środowiska Grails posiada szereg cech użytkowych, których istnienie w alternatywnych środowiskach bardzo często leżało w gestii wdrożenia ich przez samego programistę:

- uproszczony mechanizm mapowania obiektowo-relacyjnego (ORM) oparty na środowisku Hibernate;
- kontrolery zbudowane na bazie środowiska Spring MVC;
- język tworzenia widoków Groovy Server Pages (GSP);
- wbudowany kontener servletów Jetty;
- mechanizm wstrzykiwania zależności oparty na środowisku Spring Framework;
- wbudowany mechanizm umiędzynaradawiania.

Jednak ponad wszystko to właśnie wykorzystywanie języka dynamicznego Groovy jest głównym atutem środowiska Grails i wyróżnia je spośród szerokiej gamy środowisk świata Javy, tworząc intuicyjne rozwiązanie zapatrzone w język, który wyglądem i działaniem przypomina to, do czego jesteśmy już przyzwyczajeni.

Groovy

Groovy jest dynamicznym językiem obiektowym obiektowo działającym na maszynie wirtualnej Java (JVM). Pierwsze wzmianki o języku pojawiły się na blogu Jamesa Strachana w 2003 roku, zaś wersja 1.0 języka została opublikowana w 2007 roku. Groovy tworzy alternatywę dla klasycznego statycznie typowanego języka Java dzięki zaczerpnięciu cech takich języków jak: Ruby, Python, Perl, a nawet Smalltalk czy Lisp. Przykładowe zestawienie składni języków Ruby, Java oraz Groovy znajduje się w Listingach 1, 2 i 3. Mimo składni zbliżonej do wielu języków dynamicznych, Groovy potrafi z powodzeniem używać większości elementów klasycznej składni języka Java, a co za tym idzie ,umożliwia korzystanie z istniejących bibliotek języka Java. Najistotniejszą cechą języka Groovy jest fakt, że jest to język dynamiczny. Kompilatoro-

wi przetwarza kod do standardowego kodu bajtowego w trakcie wykonywania aplikacji. Czyni to z języka Groovy potężny język skryptowy posiadający dostęp do obiektów działającej aplikacji. Ale uwaga, czyni to język bardzo dynamicznym i w ten sposób to, co nie istniało podczas fazy kompilacji (i byłoby tym samym nie do zaakceptowania przez kompilator języka Java), może pojawić się później, już w trakcie działania programu w Groovy.

Programowanie w oparciu o rusztowanie

Jednym z głównych atutów środowiska Grails jest programowanie w oparciu o rusztowanie (ang. *Scaffolding*). Funkcjonalność rusztowania została zaimplementowana w wielu współczesnych środowiskach takich jak Ruby on Rails, CakePHP, Monorail, Gaia, i było niedostępne dla programistów Java aż do teraz. Scaffolding jest metodą tworzenia aplikacji, wykorzystując gotowe elementy. Elementy te stanowią wsparcie początkowych etapów projektowania i implementacji aplikacji. W miarę postępu przy tworzeniu własnych implementacji tych elementów , można bez problemu *zdjąć* wykorzystywane elementy, podobnie jak to się dzieje przy zdejmowaniu rusztowania z budynku, gdy jest on już zakończony.

Autor takiej aplikacji tworzy obiekty domenowe, na podstawie których środowisko generuje dalszy kod aplikacji. Metoda pozwala na tworzenie szybkich prototypów aplikacji zgodnych z przyjętą konwencją. Przykładowo, dla aplikacji bazujących na obiektach domenowych środowisko generuje funkcjonalność typu CRUD (ang. *Create*, *Read*, *Update*, *Delete*). Funkcjonalność ta to nic innego jak zestaw mechanizmów umożliwiających: tworzenie, czytanie, aktualizację oraz kasowanie elementów w bazie danych z poziomu aplikacji.

W zależności od przyjętego sposobu użycia mechanizmu rusztowania posiadamy możliwość modyfikacji stworzonych elementów lub nadpisania ich swoimi. Napisana w ten sposób aplikacja może zostać w pewnym etapie pozbawiona rusztowania, gdy wszystkie wymagane funkcje będą oprogramowane przez nas.

Przykładowo, załóżmy że chcemy stworzyć prostą aplikację o wspomnianym typie CRUD. W środowisku Grails sprowadza się to do stworzenia klasy domenowej, np. Samochód, gdzie podane zostaną właściwości klasy, np. kolor, data produkcji, marka. Następnie należy wybrać tryb rusztowania, w środowisku Grails do dyspozycji mamy rusztowanie dynamiczne i statyczne.

Przy dynamicznym rusztowaniu generowany kod reaguje na każdą zmianę w klasie domenowej. Przykładowo, dodając właściwość przebieg do naszej klasy Samochód, środowisko automatycznie uzupełni widoki o odpowiednie pole i umożliwi kontrolerowi edycję oraz sortowanie po tym polu. Dynamiczne rusztowanie w środowisku Grails ma dużą zaletę, którą jest możliwość nadpisywania elementów rusztowania. Przykładowo, jeśli nie podoba nam się sposób wyświetlania listy samochodów, można z powodzeniem stworzyć własny plik *list.gps*. Plik obsłuży wyświetlanie listy przy zachowaniu całej reszty rusztowania bez zmian.

Statyczne podejście polega na generowaniu rzeczywistych klas i plików widoków rusztowania. Statyczne rusztowanie możemy wygenerować, wykonując polecenie grails generate-all lub używając narzędzia np. NetBeans IDE. Spowoduje to, iż kod zostanie zapisany w postaci źródeł na dysku i wszelkie zmiany będą musiały być wprowadzone przez nas ręcznie. Jest to uciążliwe, gdy bardzo często zmieniamy klasy domenowe. Jednak pozwala to na modyfikację kodu używanego rusztowania.

Niezależnie od wybranego trybu środowisko wygeneruje odpowiednie metody dla kontrolera do obsługi klasy Samochód, oraz zestaw widoków. Stworzona w ten sposób aplikacja posiada implementację wszelkich operacji niezbędnych do zarządzania zbiorem książek w bazie danych.

Cała potęga mechanizmu rusztowania polega na szybkim wytwarzaniu prototypu, który można potem dowolnie modelować, dodając nowe lub modyfikując domyślne operacje. W naszym przypadku operacje stworzone przez rusztowanie to: tworzenie książki, wyświetlanie danych o książce, wyświetlanie listy książek, aktualizacja danych o książce i kasowanie wybranej książki.

Struktura aplikacji

Wiemy już, czym jest środowisko Grails, Groovy i Scaffolding. Czas więc zabrać się za praktyczne wykorzystanie tej wiedzy i stworzenie aplikacji. Rozpoczęcie pracy ze środowiskiem Grails zaczynamy od tworzenia struktury projektu. Najprostszym i zalecanym sposobem na stworzenie struktury jest wywołanie polecenia grails `create-app`. Jako parametr należy podać nazwę nowej aplikacji. W naszym przypadku będzie to nazwa programu do zarządzania blogiem – GrailsBlog. Wynik działania polecenia przedstawiony jest na Rysunku

Listing 1. *Przykładowa metoda w języku Ruby*

```ruby
def myRailsMethod
    i = 10
    while i < 0
        @article = Article.new
        @article.save
        i = i + 1
    end
end
```

Listing 2. *Przykładowa metoda w języku Java*

```java
public void myJavaMethod() {
    int i = 10;
    while ( i < 0 ) {
        Article article = new Article()
        article.save();
        i = i + 1;
    }
}
```

Listing 3. *Przykładowa metoda w języku Groovy*

```groovy
def myGroovyMethod = {
    def i = 10
    while ( i < 0 ) {
        def article = new Article()
        article.save()
        i = i + 1
    }
}
```

Najważniejsze polecenia środowiska Grails

- `grails clean` – czyści katalog tymczasowy środowiska;
- `grails create-app` – tworzy strukturę aplikacji opartej na środowisku Grails;
- `grails create-controller` – generuje pustą klasę kontrolera;
- `grails create-domain-class` – generuje pustą klasę domenową;
- `grails create-taglib` – tworzy pusty znacznik dla widoku;
- `grails generate-controller` – generuje klasę kontrolera na podstawie klas domenowych;
- `grails generate-views` – generuje pliki **.gsp* na podstawie klas domenowych;
- `grails generate-all` – generuje kontrolery i widoki na podstawie klas domenowych;
- `grails run-app` – uruchamia aplikacje, używając kontenera Jetty na porcie 8080;
- `grails run-app [-https]` – uruchamia aplikacje, używając protokołu https na porcie 8443;
- `grails test-app` – uruchamia testy jednostkowe w aplikacji;
- `grails war` – tworzy archiwum WAR dla kontenerów zgodnych z Java EE.

1. Struktura aplikacji pisanej w środowisku Grails oparta jest o konwencję. Oznacza to, że każdy z katalogów tej struktury ma zdefiniowaną funkcję.

Najbardziej istotnym katalogiem aplikacji jest katalog grails-app, w którym znajdują się najważniejsze elementy funkcjonalne naszej aplikacji. Zawiera on kilka istotnych podkatalogów, takich jak: *conf*, *controllers*, *domain*, *i18n*, *services*, *taglib*, *utils* i *views*. W katalogach *domain*, *controllers* i *views* znajdują się odpowiednio klasy domenowe, kontrolery oraz pliki widoków GSP. Katalog *conf* zawiera pliki i katalogi niezbędne do konfiguracji środowiska. Katalog *services* zawiera klasy usługowe, czyli niezależną logikę biznesową wywoływaną przez kontroler. Katalog taglib zawiera definicje własnych znaczników używanych na stronach GSP. Ostatecznie katalog *i18n* zawiera tzw. *message bungles*, czyli pliki zawierające teksty aplikacji w różnych językach. Dla każdego, kto miał chociaż trochę wspólnego z projektami Java EE, dalsza struktura wydaje się być logiczna i przejrzysta. Na uwagę zasługuje katalog test, gdzie gromadzone są testy jednostkowe oraz integracyjne. Środowisko posiada wbudowany mechanizm tworzenia testów. Dzięki temu testowanie jest przyjemne dla programisty. Nic tak nie przemawia do programisty jak samogenerujące się testy.

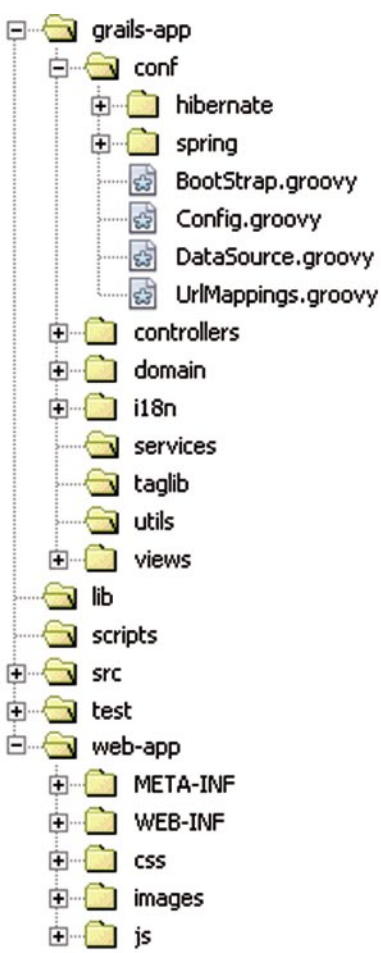
Konfiguracja

Przed rozpoczęciem pisania właściwej aplikacji rozsądnym podejściem jest przyjrzenie się konfiguracji środowiska. Dużym plusem dla środowiska i jego autorów jest fakt, iż domyślna konfiguracja jest w stanie zaspokoić wszystkie nasze początkowe potrzeby i specjalnie nie trzeba jej zmieniać. Ma to duży wpływ na odbiór środowiska przez programistę, który, pisząc kod, od razu może obserwować jego działanie, bez potrzeby analizy dokumentacji.

Konfiguracja środowiska Grails została tak rozwiązana, aby możliwe było pozbycie się tzw. koszmaru XML. Po prostu nie piszemy plików XML, tylko klasy lub skrypty w języku Groovy, dzięki czemu cała konfiguracja jest intuicyjna, czytelna i ma postać kodu – do którego przecież programiści są przyzwyczajeni. Nie uwzględniając dodatkowych wtyczek oraz integracji ze środowiskiem Spring i Hibernate, całość konfiguracji aplikacji można zawrzeć w czterech plikach. Są to:

- *Config.groovy*;
- *DataSource.groovy*;
- *UrlMappings.groovy*;
- *BootStrap.groovy*.

W pliku *Config.groovy* zawarta jest główna konfiguracja aplikacji. Należy przez to



Rysunek 1. *Struktura aplikacji działającej w środowisku Grails*

Listing 4. *Przykładowa konfiguracja źródła danych*

```
enviroment {
   development {
      dataSource {
         dbCreate = „create-drop"
         url = „jdbc:hsql:mem:blogDB"
         username = „sa"
      }
   test {
      dataSource {
         dbCreate = „update"
         url = „jdbc:mysql://localhost/testBlogDB"
         driverClassName = „com.mysql.jdbc.Driver"
         username = „tester_no_23"
         password = „sometestpassword"
      }
   }
   production {
      dataSource {
         dbCreate = „update"
         url = „jdbc:mysql://blogDBHost/blogDB"
         driverClassName = „com.mysql.jdbc.Driver"
         username = „root"
         password = „somepassword"
      }
   }
}
```

Listing 5. *Zmiana domyślnego mapowania*

```
class User {
   // ...
   static mapping = {
   table 'uzytkownicy'
   version false
   id column: 'id_uzytkownika'
      columns {
      name column: 'pierwsze_imie'
      // ...
      }
   }
}
```

rozumieć, iż plik ten traktowany jest jako główne źródło stałych konfiguracji wykorzystywanych przez aplikację. Domyślnie konfiguracja główna składa się z definicji typów plików akceptowanych przez kontener, używanych kodowań oraz podstawowej konfiguracji mechanizmu logowania. Jednak nic nie stoi na przeszkodzie dla dodania własnych stałych, które następnie możemy używać w aplikacji. Przykładowo załóżmy, że chcemy ustawić wyświetlaną wersję naszej aplikacji, stwórzmy więc wpis: `blogApp.version="Grooviest Blog Ever 1.0 google-like Beta"`. Następnie w dowolnym miejscu aplikacji za pomocą obiektu `GrailsApplication` pobieramy tę wartość. Przykładowy odczyt zmiennej wygląda następująco: `def version = grailsApplication.config.blogApp.version`. Istotną cechą konfiguracji w środowisku Grails jest to, iż mamy możliwość tworzenia stałych w postaci wpisu do klasy języka Groovy. Tak więc nie musimy konwertować wszystkiego z typu `String`, ponieważ tworzymy stałe od razu określonego typu. Jeśli jednak wolimy klasyczną konfigurację w stylu plików typu properties lub pragniemy przenieść pewne elementy konfiguracji do innych plików groovy, to również jest to możliwe. Wystarczy tylko dodać własny wpis na temat lokalizacji zewnętrznej konfiguracji do właściwości grails.config.locations w głównym pliku konfiguracyjnym.

Plik *DataSource.groovy*, jak sama nazwa mówi, zawiera informacje o konfiguracji źródeł danych dla aplikacji. Silnik Hibernate przetwarza pośrednio tę konfigurację do postaci obiektu *javax.sql.DataSource*, który następnie tworzy połączenia. Przewagą tworzenia konfiguracji źródła danych w klasie aplikacji jest brak obowiązku tworzenia źródeł na każdej instancji serwera aplikacyjnego. Nie wspominając już o uwolnieniu się od konfiguracji pliku XML. Domyślna konfiguracja zawiera jedno źródło danych wskazujące na bazę HSQLDB. Jednak nic nie stoi na przeszkodzie, aby zmienić rodzaj bazy. Dostępne są wszystkie bazy wspierane przez środowisko Hibernate. Każdy, kto kiedykolwiek miał do czynienia ze środowiskiem Hibernate, zauważy, iż plik *DataSource.groovy* posiada identyczne pola co klasyczna konfiguracja tego popularnego środowiska. Dodatkowym atutem źródeł danych w środowisku Grails jest możliwość definiowania źródła w zależności od trybu pracy. Na Listingu 4 widzimy przykładową konfigurację do tworzonej przez nas aplikacji. Konfiguracja źródeł jest podzielona na trzy segmenty: development, test i production. W zależności od trybu uruchomienia aplikacji środowisko będzie korzystać tylko z jednego segmentu, ułatwiając przy tym życie testerom i programistom .

Plik UrlMappings.groovy zawiera konfigurację kontenera odpowiedzialną za odwzorowanie adresów na odpowiednie akcje kontrolerów. Konfiguracja ma postać zwykłej klasy języka Groovy. Dzięki intuicyjnej konstrukcji pozwala na proste modyfikacje domyślnej metody mapowania, która wygląda następująco: `/$controller/$action?/$id?`. Wpis oznacza, iż przykładowo wywołanie akcji list z kontrolera EntryController odbywać się będzie, gdy klient zażąda odpowiedzi z kontekstu `/entry/list`. Konfigurację można zmieniać w dowolny rozsądny sposób, mechanizm daje nam duże pole do popisu. Dodatkowo ilość przykładów takiej konfiguracji zlokalizowanej w Internecie jest naprawdę duża.

Ostatnim omawianym plikiem związanym z konfiguracją środowiska Grails jest plik BootStrap.groovy. Z praktycznego punktu widzenia zawiera on informacje o wykonywanych podczas startu aplikacji. Przykładowo, jeśli pragniemy, aby nasza aplikacja posiadała jakieś dane po uruchomieniu, to piszemy odpowiedni kod w tym właśnie pliku. Na tym kończymy naszą przygodę z konfiguracją środowiska. Było prosto, szybko oraz bez ani jednej linii XML-a. Czas więc zabrać się za właściwą aplikację.

Klasy dziedzinowe

Konstrukcja naszej aplikacji blogowej zacznie się od wyznaczania dziedziny, w której będzie pracować. Znaczy to nie więcej niż określenie, na jakich klasach/obiektach domenowych będziemy opierać działanie programu. Dla celów prezentacji działania środowiska ograniczymy się do tworzenia trzech klas dziedzinowych. Powinny w zupełności wystarczyć do stworzenia działającego bloga.
Tymi klasami są:

- `Entry` – reprezentująca wpis w blogu;
- `Comment` – reprezentująca komentarz do wpisu;
- `User` – typowa encja pozwalająca na identyfikacje użytkownika.

Wszystkie trzy klasy domenowe można stworzyć ręcznie lub za pomocą polecenia grails `create-domain-class [Nazwa_Klasy]`. Polecenie wygeneruje plik o rozszerzeniu *groovy*, np. *User.groovy*. Wygenerowany przez nas plik jest pełnoprawną klasą języka Groovy. Plik będzie zlokalizowany w katalogu *grails-app/domain*. Wygenerowane w ten sposób trzy klasy nie będą miały żadnej implementacji z wyjątkiem bloku *static constraints*. Blok constraints zawiera metadane o naszej klasie, takie jak na przykład: kolejność wyświetlania przy użyciu rusztowania, listę możliwych stanów danej zmiennej, podstawowe informacje potrzebne przy kontroli poprawności. Dla klasy `User` możemy użyć, jeśli będziemy posiadać odpowiednie właściwości w tej klasie, następujących elementów blogu `static constraints`:

- `login(size: 5..40, blank: false, unique: true)`
- `password(blank:false, password: true)`
- `userType(inList: ["User","Admin"])`

W ten sposób poinformujemy środowisko Grails, że przy wykorzystanym mechaniźmie dynamicznego rusztowania:

Listing 6. *Klasa domenowa User*

```
class User {
    String login
    String password
    String name
    String surname
    Boolean userType
    static constraints = {
        login(size: 1..40, blank:
                    false, unique:
                    true)
        password(blank:false,
          password:true)
        name()
        surname()
        userType(inList:
          ["User","Admin"])
    }
}
```

Listing 7. *Klasa domenowa Entry*

```
class Entry {
    String title
    String body
    static hasMany = [comments:
                    Comment]
    static constraints = {
    }
}
```

Listing 8. *Klasa domenowa Comment*

```
class Comment {
    String author
    String body
    Entry entry
    static belongsTo = [entry:Entry]
    static constraints = {
    }
}
```

- kolejność wyświetlania właściwości klasy `User` to `login`, `password`, `userType`;
- właściwość `login` oraz `password` nie może być pusta;
- właściwość `login` musi być unikalna oraz mieć długość od 5 do 40 znaków;
- właściwość `password` jest typu `hasło`;
- właściwość `userType` posiada tylko dwa stany określone w parametrze `inList`.

Pozostałe właściwości tej, oraz reszty klas domenowych, znajdują się w Listingach 6, 7 oraz 8. Przeglądając listingi, możemy natrafić na pola będące metadanymi, ale nie leżącymi w bloku `static constraints`. Jak możemy się domyśleć, są to metadane określające relacje klas domenowych. Całość mechanizmu tworzenia modelu domenowego w środowisku Grails opiera się na tzn. GORM-ie (ang. *Grails Object Relational Mapping*). Jest to nakładka na środowisko Hibernate umożliwiające nam definiowanie modelu danych w klasach języka Groovy.

Mechanizm GORM posiada jedną dodatkową funkcjonalność, o której warto wspomnieć mimo faktu, iż nie będzie wykorzystywana w naszej aplikacji.Chodzi o funkcjonalność o nazwie GORM DSL (ang. *Grails Object Relationa Mapping Domail Specific Language*). Umożliwia ona korzystanie z istniejących schematów baz danych i zmiany domyślnej konfiguracji środowiska Hibernate. Dzięki niej możemy dostosować nasz model domenowy do istniejących już schematów. Przykładowe użycie języka DSL możemy zobaczyć w Listingu 5. Użyte tam mapowanie pozwala na zdefiniowanie klasy domenowej `User`, która tak naprawdę będzie tworzona na podstawie danych z tabeli o zupełnie innej konstrukcji niż definicja klasy.

Łatwo się domyśleć, iż faktyczna aplikacja będzie miała znacznie bardzie skomplikowaną warstwę danych niż ta, którą teraz implementujemy. Przedstawiony model jest bardzo uproszczony i ma za zadanie tylko przedstawić istotę działania tego elementu środowiska. Widzimy jednak, iż wraz ze środowiskiem otrzymujemy potężny zestaw mechanizmów do definiowania modelu domenowego aplikacji nawet na podstawie istniejącej już bazy danych.

Listing 9. *Klasa kontrolera użytkownika*

```
class UserController {

    def scaffold = User
    def login = {
    }

    def doLogin = {
        def user = User.findWhere(login:params['login'],password:
                        params['password'])
        if(user){
            session.user = user
            redirect(controller:'entry', action:'list')
        }else{
            redirect(controller:'user', action:'login')
        }
    }

    def logout = {
        session.user = null
        redirect(controller:'entry', action:'list')
    }
}
```

Listing 10. *Klasa kontrolera wpisu*

```
class EntryController {

    def scaffold = Entry

    def beforeInterceptor = [action:this.&checkUser, except:['index','list']]

    def checkUser = {
        if(!session.user){
            redirect(controller:'user', action:'login')
        }
    }
}
```

Listing 11. *Klasa kontrolera komentarza*

```
class CommentController {
    def scaffold = Comment

    def beforeInterceptor = [action:this.&checkUser]

    def checkUser = {
        if(!session.user){
            redirect(controller:'user', action:'login')
        }
    }
}
```

Kontrolery w blogu

Posiadając prawidłowy model domenowy naszej aplikacji, możemy się zabrać do tworzenia kontrolerów. Kontrolery w środowisku Grails są typowymi klasami języka Groovy, których pliki kończą się ciągiem znaków: *Controller.groovy*. Niech kontroler zarządzający wpisami w blogu nazywa się *EntryController.groovy*. Kontrolery są prototypowane (tworzone oddzielnie dla każdego żądania), dzięki czemu nie ma problemów z wieloma wątkami czy też kwestii spornych związanych z wzorcem singleton.

Zasada działania jest prosta, każdy kontroler posiada jedną lub więcej bloków przetwarzających zapytanie klienta. Bloki nazywane są akcjami i domyślnie reprezentują wywołanie odpowiedniego adresu przez przeglądarkę (dzięki mapowaniu zdefiniowanemu w *UrlMappings.groovy*). Przykładowo, akcja w kontrolerze *EntryController* o definicji:

```
def index = {
   render 'Boo!'
}
```

będzie wywoływana jako domyślna akcja i po przetworzeniu żądania wypisze na stro-

nie wskazany ciąg znaków. Wywołanie tej akcji może się odbyć zarówno z kontekstu `/entry/index`, jak i z kontekstu `/entry/`.

Aby stworzyć kontroler, wystarczy wywołać polecenie grails `create-controller [Nazwa_Kontrolera]`. Zacznijmy od kontrolera dla klasy domenowej `User`. Kontroler zgodnie z konwencją będzie się nazywał `UserController.groovy` i będzie zlokalizowany w katalogu *controllers*. Ponieważ pragniemy mieć możliwość tworzenia, edycji, przeglądania oraz kasowania użytkowników naszego bloga, to musimy albo zaimplementować każdą z akcji samodzielnie, albo użyć mechanizmu rusztowania. Wspomniany wcześniej mechanizm w tym przypadku jest nieoceniony. Wystarczy dodać do klasy kontrolera wpis: `def scaffold = User`, aby środowisko w czasie wykonywania (dynamicznie) stworzyło wszystkie akcje operujące na klasie domenowej `User`. W konsekwencji posiadamy klasę z całkiem przyzwoitą implementacją takich akcji jak: `list`, `show`, `edit`, `delete`, `create`, `save` oraz `update`.

Jednak zarządzanie klasą domenową to nie wszystko. Użytkownik musi się przecież logować, aby mieć możliwość zarządzania blogiem. Nic więc nie stoi na przeszkodzie, aby dodać akcje odpowiedzialne za logowanie do systemu. Na Listingu 9 widzimy, że kontroler użytkownika posiada dodatkowe trzy akcje z tym związane.

Akcja `login` nie posiada implementacji, ponieważ jej działanie ma na celu wyświetlenie ekranu logowania. Zgodnie z konwencją kontener po przejściu przez akcję wyświetla stronę widoku o nazwie odpowiadającej nazwie wykonanej akcji. W tym przypadku zostanie wyświetlony plik *login.gsp*, o którym więcej dowiemy się w części artykułu poświęconej widokom.

Akcja `doLogin` zawiera implementację logiki związanej z faktycznym logowaniem się użytkownika. Akcja używa statycznej metody `findWhere` klasy domenowej `User` do znalezienia w bazie wpisu pasującego do podanego hasła oraz identyfikatora. Zauważyłeś pewnie Drogi Czytelniku fakt, iż nie pisaliśmy przecież w modelu domenowym żadnego mechanizmu wyszukiwania. Zwłaszcza że napisanie tak generycznej metody na pewno by zajmowało wiele linii kodu. W skrócie mówiąc, tworząc klasę domenową `Grails` (za pomocą Groovy), automatycznie wyposażona ona zostaje w zestaw przydatnych statycznych metod, które możemy wykorzystać w kontrolerze. Przykładowe metody to: `count`, `createCriteria`, `exists`, `find`, `get` i `list`. Środowisko Grails posiada wiele takich niespodzianek. Użyta metoda `findWhere` potrafi odszukać wpis w bazie spełniający kryteria. W przypadku gdy metoda znajdzie pasujący wpis, akcja ustawia go w zakresie sesji. Wszystko dzięki prostemu i intuicyjnemu wpisowi `session.user = user`. Ostatecznie gdy akcja ustawi już obiekt w sesji, przekierowuje zapytanie do następnej akcji metodą `redirect`. Podobne przekierowanie odbywa się w przypadku, gdy użytkownikowi nie uda się zalogować.

Ostatnim napisanym przez nas blokiem w kontrolerze `User` będzie akcja `logout`, będzie ona odpowiedzialna za wylogowanie użytkownika z bloga. Odbywa się to przez wyczyszczenie referencji do obiektu w sesji, czyli zwykłym wywołaniem `session.user = null`. Przedstawiony mechanizm daleko odbiega od ideału, ale doskonale pokazuje w skrócie możliwości kontrolerów. Jeśli jednak pragniemy napisać mechanizm autoryzacji z prawdziwego zdarzenia, to możemy skorzystać z wtyczki AcegiSecurity. Udostępnia ona mechanizm znany użytkownikom środowiska Spring.

Posiadamy już mechanizm zarządzania użytkownikami włącznie z logiką autoryzacji. Trzeba teraz to wykorzystać i napisać kontrolery do klas domenowych `Entry` i `Comment`. Ponieważ każdy programista ma

Listing 12. *Fragment strony logowania użytkownika*

```
<g:form action="doLogin" method="post">
   <div class="dialog">
      <p>Enter your login details below:</p>
      <table class="userForm">
         <tr class='prop'>
         <td valign='top' style='text-align:left;' width='20%'>
            <label for='login'>Login:</label>
         </td>
         <td valign='top' style='text-align:left;' width='80%'>
            <input id="login" type='text' name='login' value='${user?.login}' />
         </td>
         </tr>
         <tr class='prop'>
         <td valign='top' style='text-align:left;' width='20%'>
            <label for='password'>Password:</label>
         </td>
         <td valign='top' style='text-align:left;' width='80%'>
            <input id="password" type='password' name='password'
                     value='${user?.password}' />
         </td>
         </tr>
      </table>
   </div>
   <div class="buttons">
      <span class="formButton">
         <input type="submit" value="Login"></input>
      </span>
   </div>
</g:form>
```

Listing 13. *Wyświetlanie listy wpisów wraz z komentarzami*

```
<g:each in="${entryInstanceList}" status="i" var="entryInstance">
   <g:link action="show" id="${entryInstance.id}">
      ${fieldValue(bean:entryInstance, field:'title')}
   </g:link>
   ${fieldValue(bean:entryInstance, field:'body')}
   Comments:
   <g:each in="${entryInstance.comments}" status="j" var="commentInstnace">
      Author: ${fieldValue(bean:commentInstnace, field:'author')}
      ${fieldValue(bean:commentInstnace, field:'body')}
   </g:each>
</g:each>
```

W Sieci

- *http://grails.org/*
- *http://groovy.codehaus.org*

w sobie coś z leniucha, to także i my ułatwimy sobie trochę życie, korzystając z dostarczanych przez Grails funkcjonalności. Użyjemy mechanizmu rusztowania także i w tych dwóch kontrolerach. Jak widzimy w Listingach 10 oraz 11, wpisy informujące o używaniu rusztowania różnią się tylko typem klasy domenowej używanej przez kontroler. Dodatkowo w obydwu kontrolerach zaimplementowana jest prosta akcja sprawdzająca, czy w sesji istnieje już zalogowany użytkownik. Jeśli nie, akcja przekierowuje do żądania do akcji logowania. O ile konstrukcja tej akcji jest dosyć prosta, o tyle jej wykorzystanie już jest bardziej interesujące.

Zauważyłeś pewnie Drogi Czytelniku, iż w każdym z omawianych kontrolerów istnieje zmienna `beforeInterceptor`. Przypisana do niej akcja będzie wykonywana zawsze przed wywołaniem akcji docelowej. W konsekwencji, jeśli użytkownik będzie chciał na przykład dodać wpis do bloga, a nie będzie zalogowany, to zostanie od razu przekierowany do ekranu logowania. Dodatkowo przy deklaracji zmiennej możemy użyć parametru except, zawierającego listę akcji, przy których nie uwzględnia się wywołania akcji `checkUser`. Ostatnią rzeczą wartą wyjaśnienia przy tworzeniu kontrolerów jest sposób wywołania akcji przy użyciu przedrostka `&`. Język Groovy pozwala na bezpośrednie oraz pośrednie odwoływanie się do pól oraz metod. Aby wykorzystać sposób bezpośredni przy metodzie, należy użyć omawianego przedrostka. W ten sposób interpreter języka odwoła się do metody (akcji) bez pośrednictwa innych niepotrzebnych metod.

Widoki

Środowisko Grails udostępnia własny mechanizm tworzenia widoków w postaci plików o rozszerzeniu gsp (ang. *Groovy Server Pages*).Jest on bardzo podobny do klasycznego Java Server Pages, więc każdy, nawet początkujący programista, powinien od razu się odnaleźć w nowych znacznikach. Widok tworzy się przy użyciu zestawu dodatkowych znaczników, domyślnie zaczynających się od przedrostka g, przykładowo g: form, wymieszanych na przemian z tradycyjnymi znacznikami języka HTML. Technologia dodatkowo GSP dostarcza znacznie bardziej rozszerzony język wyrażeń (ang. *Expression Language*) w porównaniu z klasyczną biblioteką JSTL.

Na Listingu 12 widzimy prosty formularz logowania, który będzie znajdować się we wspomnianej już wcześniej stronie *login.gsp*. Zapewni nam on możliwość wywołania akcji `doLogin` z poziomu ekranu logowania. W konsekwencji pozwoli nam on zalogować się do naszego bloga. Formularz zawiera tylko jeden znacznik wychodzący poza listę znaczników języka HTML. Tworząc stronę w technologii GSP, nie musimy stosować skomplikowanych znaczników zastępujących znaczniki input, jak to ma miejsce w środowisku JSF. Możemy zauważyć, iż przy polach tekstowych w atrybucie `value` zastosowana została konstrukcja zawierająca znak `?`. Znak ten jest operatorem bezpiecznej dereferencji (ang. *safe dereference operator*). Dzięki niemu, w sytuacji gdy z jakichkolwiek przyczyn w zakresie strony brakować będzie obiektu `user`, to użytkownik nie otrzyma informacji o wyjątku. W tym przypadku wyrażenie będzie miało wartość `null`. W efekcie oznacza to, iż będzie ono traktowane jako pusty ciąg znaków.

Posiadamy już ekran logowania. Cała reszta widoków w aplikacji w zasadzie wygenerowana została automatycznie. Jedynie część wyświetlającą listę wpisów na blogu musimy tak zmodyfikować, aby ładnie pokazywała treść bloga i komentarze. Przez określenie *ładnie* mam na myśli tylko i wyłącznie piękno kodu źródłowego. Stroną wizualną omawianego ekranu proponuję zająć się we własnym zakresie. Dlatego też w Listingu 13 widzimy znaczniki ze strony *list.gsp* kontrolera `entry` pozbawione wszelkich wpisów klasycznego języka HTML. Przedstawiona logika prezentacji strony składa się z dwóch pętli. Pętla zewnętrzna operuje na wpisach, wypisując tytuł wpisu oraz ciało wpisów. Druga wewnętrzna pętla iteruje po wszystkich komentarzach danego wpisu i wyświetla autora komentarza oraz jego treść. W przedstawionym listingu całość kodu wygląda przyjemnie i przede wszystkim ładnie. Osoba znająca bibliotekę JSTL powinna widzieć wyraźne podobieństwa i z łatwością powinna się oswoić z nową technologią.

Działający kod wraz z odpowiednimi znacznikami HTML i stylami przedstawiony jest na Rysunku 2.

Rysunek 2. *Widok listy wpisów z uwzględnionymi znacznikami HTML*

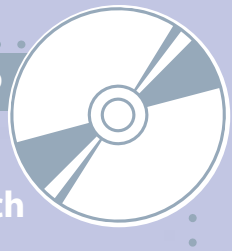
Podsumowanie

Początkowo podszedłem do środowiska Grails bardzo sceptycznie. Można nawet powiedzieć, że z góry skazałem tę technologię na przegraną. Moje opinia była oparta na fakcie, iż z roku na rok wychodzi nowe środowisko lub nowa *ładniejsza* wersja już istniejącego, oraz fakcie, iż przy tym konkretnym środowisku trzeba się nauczyć nowego języka. Jednak spróbowałem stworzyć prostą aplikację, aby zobaczyć chociaż, czym niektórzy programiści się zachwycają. Zmieniłem moją negatywną opinię, gdy po 90 minutach udało mi się dodać oraz modyfikować elementy w bazie danych przez stworzoną aplikację. Muszę zaznaczyć, że przedtem nie wiedziałem praktycznie nic o środowisku, więc ilość czasu, którą na to poświęciłem, naprawdę mnie zaskoczył. Nie przekonuję Cię Drogi Czytelniku, abyś polubił to środowisko, ale proponuję: spróbuj napisać chociaż jedną aplikację, a na pewno wyrobisz sobie pozytywną opinię o tej technologii. Zwyczajnie spróbuj pobić mój czas, po tej lekturze na pewno Ci się to uda.

MARIUSZ RÓG

Pracuje na stanowisku Solution Developer w firmie Acxiom Polska. Specjalizuje się w tworzeniu aplikacji wykorzystujących złożone zasoby danych dla międzynarodowych klientów wielu sektorów przemysłu. Acxiom Polska jest częścią notowanej na giełdzie NASDAQ globalnej korporacji Acxiom. Łącząc dane, unikalne know-how oraz technologię, zapewnia kompleksowe rozwiązania w zakresie zarządzania zintegrowanymi informacjami o klientach.

Kontakt z autorem: mariusz.rog@gmail.com

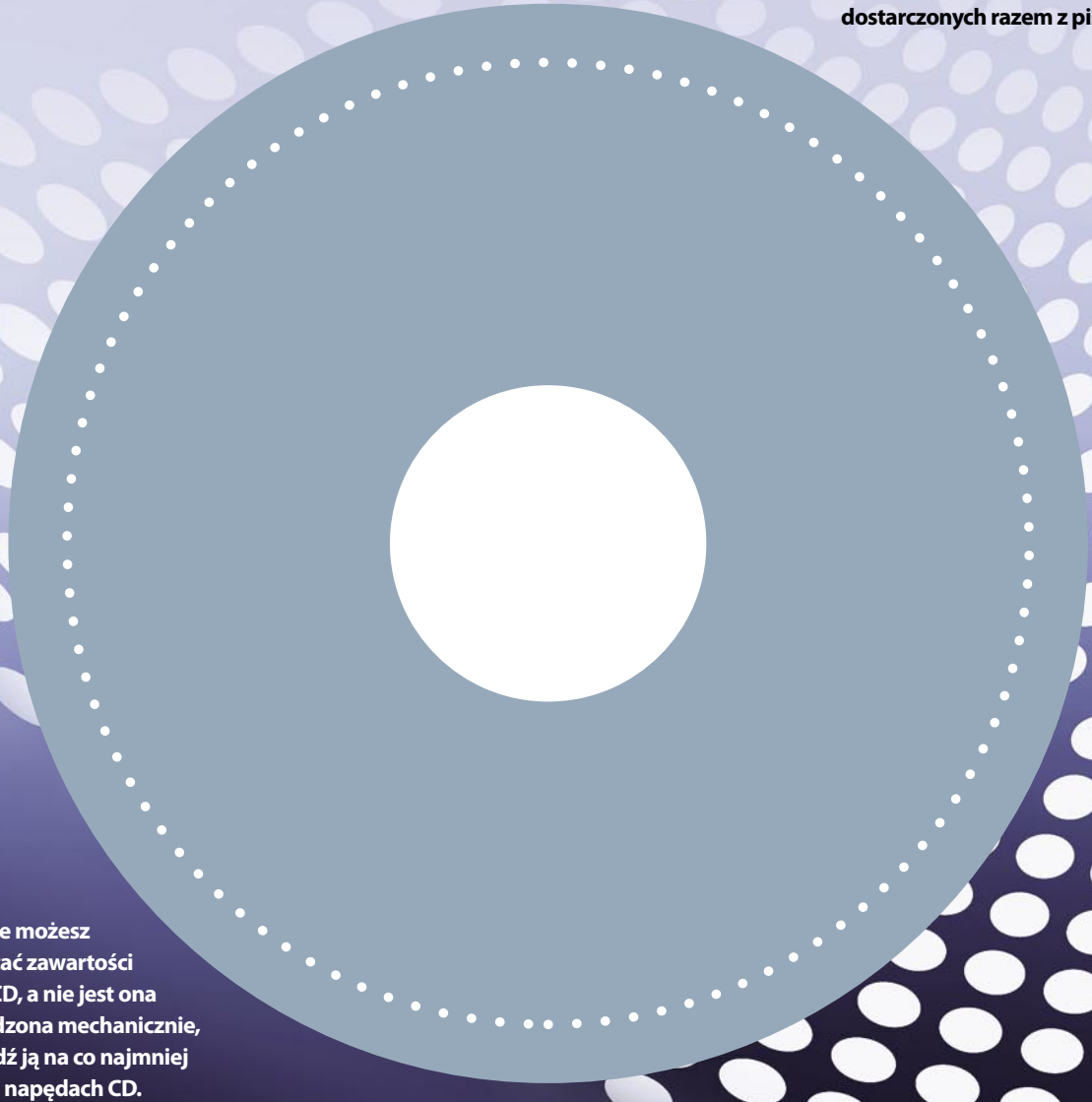


Redakcja Software Developer's Journal specjalnie dla swoich najwierniejszych czytelników przygotowała archiwum wszystkich wydań SDJ i SDJ Extra z 2008 roku.

Nasi czytelnicy mogą mieć dostęp do wszystkich artykułów w formacie .pdf. Elektroniczna wersja wszystkich artykułów z 2008 roku to egzemplarz unikatowy i każdy powinien go mieć!

Software Developer's JOURNAL
new ideas & solutions for professional programmers

Redakcja nie udziela pomocy technicznej w instalowaniu i użytkowaniu programów zamieszczonych na płytach CD-ROM dostarczonych razem z pismem.



Jeśli nie możesz odczytać zawartości płyty CD, a nie jest ona uszkodzona mechanicznie, sprawdź ją na co najmniej dwóch napędach CD. W razie problemów z płytą, prosimy pisać pod adres: *cd@software.com.pl*

Bazy danych w Adobe AIR

Adobe AIR i SQLite. Zapoznajemy się z wbudowaną w runtime AIR obsługą bazy danych.

Dowiesz się:
- Jak wykorzystać bazę danych w AIR

Powinieneś wiedzieć:
- Podstawy technologii Flex i AIR
- Podstawy języka ActionScript 3
- Jak posługiwać się jezykiem SQL

Poziom trudności

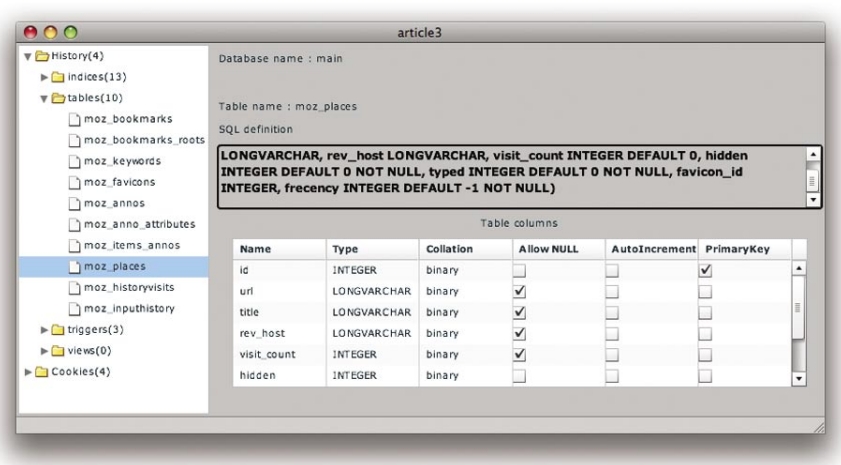
Jedną z ważnych cech Adobe AIR, nie poruszaną przez nas do tej pory, jest obsługa bazy danych. W aktualnej wersji AIR daje możliwość korzystania z wbudowanej obsługi bazy danych SQLite. Dzięki temu możemy korzystać z prawie wszystkich cech języka SQL92. Krótka lista nie obsługiwanych elementów standardu dostępna jest na stronie projektu SQLite. Warto tutaj wspomnieć, że SQLite implementuje między innymi podzapytania, triggery czy też widoki. Dodatkowo w przypadku SQLite nie mamy osobnego procesu RDBMS, cała baza danych przechowywana jest w pojedynczym pliku. Połączenie z bazą sprowadza się zatem do wybrania i otwarcia odpowiedniego pliku. Pliki z bazą danych nie są standardowo kodowane, dzięki czemu możemy bazą zarządzać korzystając z dostarczanej aplikacji konsolowej.

SQLite jest bardzo popularnym systemem zarządzana bazą danych. Jest między innymi używany przez przeglądarkę Mozilla Firefox. Baza danych została zastosowana do przechowywania historii odwiedzanych stron internetowych, historii pobieranych plików, cookies czy też zapisanych zakładek. Ponieważ pliki baz danych nie są szyfrowane możemy odczytać je w naszej aplikacji AIR. W pierwszej jednak kolejności zapoznajmy się z podstawową obsługą bazy danych w Adobe AIR.

Połączenie z bazą danych

Zarządzanie bazą danych realizowane jest poprzez klasy z pakietu *flash.data.**, pakiet ten jest dostępny jedynie dla aplikacji AIR. Do połączenia z bazą danych wykorzystujemy klasę `flash.data.SQLConnection`. Przykładowe nawiązanie połączenia z bazą danych przedstawione zostało na Listingu 1. Prezentowany kod ilustruje sposób połączenie z bazą danych przechowywaną w pliku *artTest1.db*. W pierwszym kroku tworzymy obiekt odpowiedzialny za połączenie z bazą. W omawianym przykładzie, jeśli plik (baza) nie istnieje, jest on tworzony w czasie wykonania metody open klasy `SQLConnection`. Wykonanie pojedynczego zapytania realizowane jest przez klasę `SQLStatement`. Wszelkie dane zwrócone w wyniku wykonania zapytania są przechowywane w obiekcie klasy `SQLResult`. Dostęp do wyników zapytania uzyskujemy poprzez metodę `SQLStatement.getResult()`. Pozyskany obiekt zawiera dane wybrane zapytaniem `SELECT (SQLResult.data)`, id dodanego wiersza dla zapytania `INSERT (SQLResult.lastInsertRowID)` oraz informacje o liczbie wierszy przetworzonych w ostatnim zapytaniu `INSERT`, `UPDATE`, `DELETE (SQLResult.rowsAffected)`.

Adobe AIR dostarcza dwa możliwe typy połączenia – synchroniczne i asynchroniczne. Połączenie synchroniczne przed-



Rysunek 1. *Struktura bazy (SQLSchema) dla bazy firefoxa*

stawione zostało na Listingu 1. Połączenia synchroniczne są łatwiejsze do zaprogramowania. Jednak, ponieważ wszystkie wywołania wykonywane są w głównym wątku, możliwe jest chwilowe zawieszenie aplikacji. Wykorzystanie połączenia asynchronicznego powoduje, że wszystkie operacje wykonywane są w trybie nie blokującym działania naszej aplikacji. Listing 2 przedstawia asynchroniczną wersję kodu zawartego w poprzednim przykładzie.

Korzystając z asynchronicznego połączenia z bazą danych, możemy wybrać sposób w jaki jesteśmy powiadamiani o zakończeniu operacji. Na Listingu 2 wykorzystujemy mechanizm *responderów*.

Obiekt klasy `Responder` posiada jedynie dwie właściwości `Responder.result` oraz `Responder.status`.

Właściwości te przechowują funkcje użytkownika, które zostaną wykonane w zależnie od tego czy wywołanie metody, dla której określony został responeder, zakończy się sukcesem czy błędem.

Obiekt klasy `Responder` może być przekazany jako parametr do większości metod klasy `SQLConnection` i `SQLStatement`. Drugie podejście polega na nasłuchiwaniu odpowiednich zdarzeń klasy `SQLConnection`. Podejście to jednak wydaje się być jednak trudniejszym do nadzorowania. Nie możemy jednak korzystać jednocześnie z obu mechanizmów, zastosowanie responderów powoduje zablokowanie rozgłaszania zdarzeń.

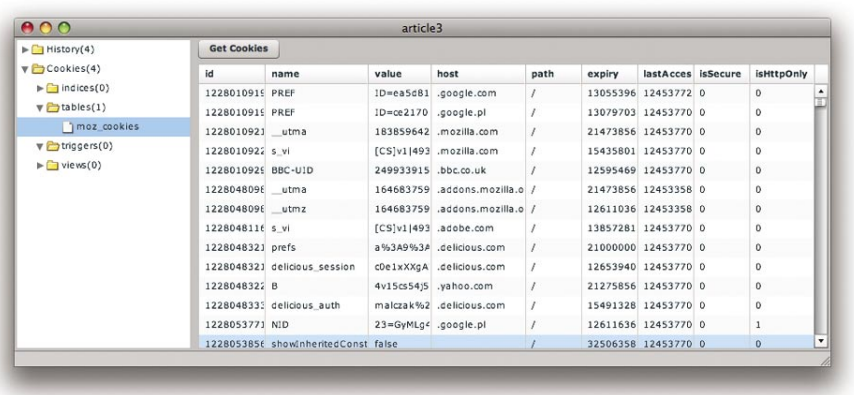
Na Listingu 2 zaprezentowana została także jeszcze jedna ważna funkcjonalność dostępna w Adobe AIR, jaką są zapytania z parametrami. Dzięki temu mechanizmowi możemy wielokrotnie wykorzystywać to samo zapytanie zmieniają jedynie wartości parametrów wstawianych do zapytania. Parametr w zapytaniu na postać `@nazwa_parametru`, aby ustawić jego wartość piszemy `query.parameters[„@nazwa_parametru”] = ‘moja wartosc’`.

Oba poprzednio przedstawione przykłady powodowały utworzenie na dysku nowej bazy danych. Jeśli jednak w aplikacji mamy potrzebę wykorzystania bazy danych, ale nie chcemy jej zapisywać na dysku, możemy utworzyć bazę w pamięci. Taka baza danych ma funkcjonalność taką samą jak baza danych przechowywana na dysku. Jednak dane w niej zapisane zostaną utracone wraz z zamknięciem połączenia z taką bazą. Wirtualne bazy danych tworzone są, jeśli jako parametr określający źródło bazy danych zamiast obiektu `File` przekazana zostanie wartość null.

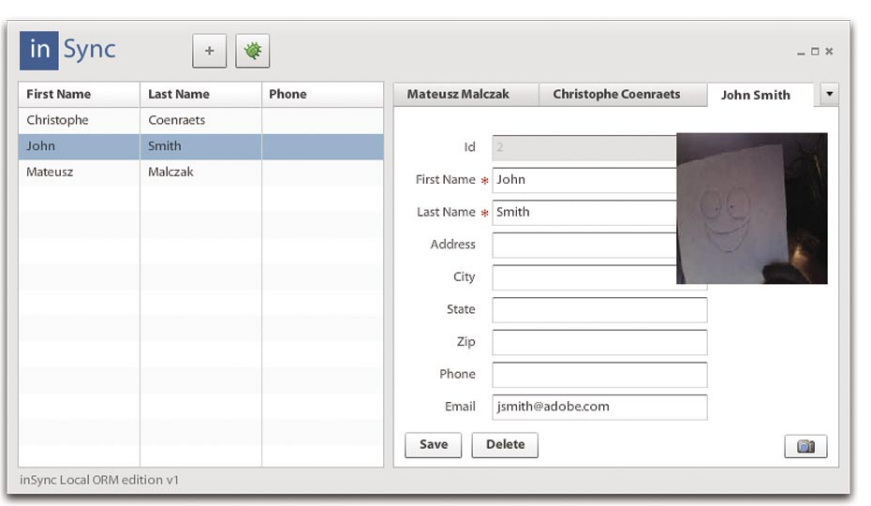
Wraz z pojawieniem się Adobe AIR 1.5 mamy także możliwość szyfrowania pliku bazy danych. Standardowo bazy danych

Listing 1. *Synchroniczne połączenie z bazą danych*

```
// tworzymy obiekt odpowiedzialny za połączenie z bazą
var sqlCon:SQLConnection = new SQLConnection();
// wybieramy bazę z którą chcemy się połączyć
var dbFile:File = File.desktopDirectory.resolvePath('artText1.db');
var firstOpen:Boolean = dbFile.exists == false;
// nawiązujemy połączenie z bazą. Jeśli baza nie istnieje zostane uwtorzona
sqlCon.open( dbFile, SQLMode.CREATE );
   // przy pierwszym otwarciu tworzymy przykładową tabele
   if ( firstOpen )
   {
      var tblc:SQLStatement = new SQLStatement();
      tblc.sqlConnection = sqlCon;
      tblc.text = "CREATE TABLE test_table ( date DATE, value float )";
      tblc.execute();
   };
// wpisujemy dane do tabeli
var query:SQLStatement = new SQLStatement();
query.sqlConnection = sqlCon;
query.text = "INSERT INTO test_table ( date, value ) VALUES ( date('now'), @VALUE
                    )";
query.parameters["@VALUE"] = Math.random();
query.execute();
// zamykamy połączenie z bazą
sqlCon.close();
```



Rysunek 2. *Cookies pobrane z bazy danych przeglądarki firefox*



Rysunek 3. *Przykładowa aplikacja inSync dostępna na stronach Adobe*

Listing 2. *Asynchroniczna wersja połączenia z Listingu 1*

```
// metoda inicjująca połączenie z bazą danych
private function sqlconnect():void
{
sqlCon = new SQLConnection();
var dbFile:File = File.desktopDirectory.resolvePath('artText1.db');
firstOpen = dbFile.exists == false;
// asynchroniczne połączenie z bazą. Jeśli wystąpi błąd wywołana zostanie metoda
                    sqlError.
// Jeśli połączenie powiedzie się wywołana zostanie metoda connectSuccess
sqlCon.openAsync( dbFile, SQLMode.CREATE, new Responder( connectSuccess, sqlError ) );
};
// po otwarciu bazy danych zainicjujemy ją lub dodamy dane
private function connectSuccess( data:Object ):void
{
   if ( firstOpen )
   {
      var tblc:SQLStatement = new SQLStatement();
      tblc.sqlConnection = sqlCon;
      tblc.text = "CREATE TABLE test_table ( date DATE, value float )";
      // zapytanie wykonane zostanie asynchronicznie
      tblc.execute( -1, new Responder( insertData, sqlError) );
   } else insertData();
};
// wstawienie danych do bazy
private function insertData( data:Object = null ):void
{
   var query:SQLStatement = new SQLStatement();
   query.sqlConnection = sqlCon;
   query.text = "INSERT INTO test_table ( date, value ) VALUES ( date('now'), @VALUE
                    )";
   query.parameters["@VALUE"] = Math.random();
   // zapytanie wykonane zostanie asynchronicznie, po nim zamykamy bazę
   query.execute( -1, new Responder( sqlclose, sqlError) );
};
// zamknięcie bazy danych
private function sqlclose( data:Object ):void
{
   sqlCon.close();
};
// powiadomienie o błędzie
private function sqlError( data:Object ):void
{
   trace( " @@ SQL ERROR @@ ");
};
```

Listing 3. *Tworzenie zaszyfrowanej bazy danych*

```
// klucz wykorzystany do zaszyfrowania bazy
var encryptKey:ByteArray = new ByteArray();
encryptKey.writeUTFBytes('superEncryptionP');
// tworzymy połączenie z nową bazą
var sqlCon:SQLConnection = new SQLConnection();
var dbFile:File = File.desktopDirectory.resolvePath('artText1.db');
var firstOpen:Boolean = dbFile.exists == false;
// tworzymy nową bazę szyfrowaną podanym wczesniej hasłem
sqlCon.open( dbFile, SQLMode.CREATE, false, 1024, encryptKey );
// Dalej jak na listingu 1
// ….
```

SQLite nie są szyfrowane, przez co każdy może naszą bazę otworzyć i zapoznać się z jej zawartością. Szyfrowanie bazy odbywa się na etapie jej tworzenia. Przykład tworzenia zaszyfrowanej bazy danych przedstawiony jest na Listingu 3.

Hasło wykorzystane do szyfrowania musi mieć długość dokładnie 16 znaków i może być zmienione w dowolnej chwili z wykorzystaniem metody `reencrypt`. Metoda ta może być wykonana jedynie na bazie wcześniej zaszyfrowanej. Nie można w ten sposób zaszyfrować bazy, która nie była wcześniej zabezpieczona. Wykorzystując szyfrowaną bazę danych oraz obszar `EncryptedLocalStore` możemy być pewni, że prywatne dane naszej aplikacji nie zostaną przechwycone.

Pobieranie struktury bazy

Jak wspomniałem we wstępie, przeglądarka Firefox korzysta z bazy SQLite do przechowywaniawielu ciekawych danych. Pobierzmy je. Swoją uwagę skupimy na plikach *cookies.sqlite* oraz *places.sqlite*. Pierwszy z nich przechowuje ciasteczka zapisane podczas odwiedzania stron. W drugim natomiast zapisana jest historia odwiedzanych stron. Jak jednak możemy pobrać dane z bazy, której struktury nie znamy ? Z pomocą przychodzi nam możliwość pobrania struktury danych (*SQLSchema*). Mechanizm ten pozwala nam pobrać informację o – tabelach, indeksach, triggerach oraz widokach zdefiniowanych w otwartej bazie danych. Dzięki temu poznajemy także zapytania SQL tworzące poszczególne elementy, oraz szczegółową charakterystykę wszystkich kolumn. Rysunek 1 przedstawia strukturę bazy przechowującej historię odwiedzanych stron z pliku *places.sqlite*. Kod realizujący to zadania przedstawiony został na Listingu 4. Dzięki możliwości pobierania sktruktury bazy możemy z łatwością czytać dowolne bazy SQLite. Rozbudowując aplikację z Rysunku 1 możemy także stworzyć użyteczny program do zarządzania bazami danych, który może być alternatywą dla aplikacji konsolowej.

Kiedy już znamy strukturę bazy, możemy przejść do pobierania danych. Na Ry-

Kody źródłowe

Kody źródłowe przykładowych aplikacji do artykułów :

- *Wprowadzenie do Adobe AIR*
- *Klient serwisu twitter.com*
- *Bazy danych w Adobe AIR*

dostępne są na stronie
http://segfaultlabs.com/docs

sunku 2 przedstawione zostały dane wyciągnięte z bazy przechowującej ciasteczka dla przeglądarki Firefox. W przykładzie tym tabela wyświetlająca dane tworzona jest dynamicznie na podstawie wcześniej pobranego opisu bazy. Następnie wykonane zostało zapytanie SELECT. Całość przedstawiona została na Listingu 5.

Listing 4. *Pobranie struktury bazy*

```
// otwieramy bazę przechowującą odwiedzone adresy url
var placesDB:File = firefoxDirectory.resolvePath('places.sqlite');
sqlCon.openAsync( placesDB, SQLMode.UPDATE, new Responder( dbGetPlacesSchema,
                        sqlError ) );

private function dbGetPlacesSchema( result:Object ):void
{
   // pobierz kompletny opis głównej bazy danych (standardowo otwarta baza nosi
                        nazę main)
   sqlCon.loadSchema( null, null, "main", true, new Responder(
                        dbPlacesSchemaLoaded, sqlError ) );
};
private function dbPlacesSchemaLoaded( result:Object ):void
{
   // wyświetl opis bazy
   placesSchema = SQLSchemaResult( result );
   displaySchema( placesSchema );
};
```

Listing 5. *Pobieranie danych z bazy*

```
private function getFirefoxCookies():void
{
   var query:SQLStatement = new SQLStatement();
   // ustawiamy uprzednio nawiązane połączenie z bazą
   query.sqlConnection = sqlCon;
   query.text = "SELECT * FROM moz_cookies";
   // pierwszy parametr oznacza, że chcemy pobrać wszystkie rekordy
   query.execute( -1, new Responder( cookiesSuccess, sqlError ) );
};
private function cookiesSuccess( data:Object ):void
{
   // tworzymy kolumny DatarGrid w których zostaną wyświtlone dane
   var cols:Array = [];
   var dataGridColumn:DataGridColumn;
   // pobieramy opis tabeli w której przechowywane są ciasteczka
   var tableSchema:SQLTableSchema = cookiesSchema.tables[0] as SQLTableSchema;
   // klasa ta opisuje postać pojedynczej kolumny w tabeli
   var colSchema:SQLColumnSchema;
      for each ( colSchema in tableSchema.columns )
      {
         dataGridColumn = new DataGridColumn( colSchema.name );
         dataGridColumn.dataField = colSchema.name;
         cols.push( dataGridColumn );
      }
   // ustawiamy kolumny
   faviconsGrid.columns = cols;
   // ustawiamy pobrane dane
   faviconsGrid.dataProvider = SQLResult(data).data;
};
```

Przykład od Adobe

Bardzo ciekawy przykład wykorzystania bazy danych znajduje się na stronach Adobe. Pod adresem *http://www.adobe.com/devnet/air/flex/articles/sqlite_db_api_in_air.html* możemy znaleźć dyskusję opisującą trzy różne podejścia do używania baz danych w aplikacji AIR. Na przykładzie aplikacji do przechowywania kontaktów, artykuł przedstawia nie opisywane tutaj podejścia *object-relational mapping* (ORM) oraz *Data Access Object* (DAO). Aplikacja ilustruje także jak można w kolumnie typu BLOB przechowywać obrazy pobrane z kamery.

Działająca aplikacja została zaprezentowana na Rysunku 3.

Podsumowanie

Artykuł ten jest jedynie wprowadzeniem do tematyki obsługi baz danych w Adobe AIR. Wykorzystanie systemu zarządzania bazą danych SQLite daje wiele możliwości. Możemy przechowywać w bazie ustawienia aplikacji, hasła użytkowników itp. Możliwość szyfrowania bazy daje nam także gwarancję bezpieczeństwa. Pliki bazy danych mogą być wykorzystywane do komunikowania się z innymi aplikacjami, serwerem czy nawet urządzeniami. Osoby które miały już styczność z SQLite szybko zauważą, że jest on w pełni wspierany w środowisku AIR. Dla osób nie mających doświadczenia z tym systemem baz danych, Adobe AIR wydaje się być odpowiednim miejscem aby rozpocząć przygodę z SQLite.

MATEUSZ MAŁCZAK

Autor jest programistą z bogatym doświadczeniem w tworzeniu aplikacji desktopowych dla systemów Windows i Linux. Aktualnie bardzo mocno związany z technologiami Flex/AIR. Tworząc aplikacje dla serwisu http://komixo.com wygrał ogólnopolski konkurs na najlepszą aplikację w tej technologii (http://www.flexchallenge.com/pl). Kontakt z autorem: matuesz@malczak.info

W Sieci

- *http://www.sqlite.org* – strona domowo projektu SQLite
- *http://livedocs.adobe.com/flex/gumbo/langref/localDatabaseSQLSupport.html* – szczegółowy opis wsparcia Adobe AIR dla SQLite
- *http://www.peterelst.com/blog/2008/12/11/flexcamp-2008-belgium/* – prezentacja oraz przykładowa aplikacja ilustrująca możliwości SQLite
- *http://coenraets.org/blog/2008/02/sqlite-admin-for-air-10* – SQLite manager napisany w Adobe AIR

Groovy

na pohybel Javie, z miłością do Javy –wprowadzenie do języka

Groovy to nowoczesny, dynamiczny język, w pełni integrujący się z platformą Java, pozwalając na bezpośrednie korzystanie z mnogości istniejących bibliotek i szkieletów.

Dowiesz się:
- Jak zacząć programowanie w języku Groovy, jaka jest jego składnia i budowa;
- Jak łączyć programy napisane w Groovim z Javą i vice versa.

Powinieneś wiedzieć:
- Jak programować w języku Java.

Poziom trudności

Dla wielu programistów, zwłaszcza tych, którzy przesiąkli pewnymi nawykami, nauka nowego języka programowania (lub choćby okazanie mu zainteresowania) jest porównywalna do zdrady partnera. Jest to niestety podejście dość radykalne i leniwe, bowiem z każdego wartościowego języka (a postaram się udowodnić, że Groovy do takich należy) można nauczyć się wiele nowego, nawet jeśli finalnie pozostanie się przy swoim ulubionym rozwiązaniu. Koncepcja ta została doskonale opisana w artykule Petera Norviga pt. *Poznaj sztukę programowania w dziesięć lat* (dostępnym na stronie internetowej *http://norvig.com/21-days.html*). Autor sugeruje, by poznać kilka różnych koncepcyjnie języków (przynajmniej jeden funkcjonalny, jeden obiektowy i równoległy) do tego stopnia, by zacząć umieć myśleć w tych językach (a nie tylko poznać ich składnię, jednak programować z użyciem dotychczasowych nawyków). Rezultatem takiego doświadczenia powinno być szersze spojrzenie na programowanie jako sztukę tworzenia kodu, z pominięciem ograniczeń narzucanych przez pojedyncze rozwiązanie. W efekcie można, w zależności od konkretnego problemu, stosować odpowiednie techniki lub łączyć ideologie. Z własnego doświadczenia wiem, że koncepcja ta jest ze wszech miar słuszna. Groovy jest bardzo ciekawym pomysłem na rozszerzenie wiedzy o językach dynamicznych i funkcyjnych (zakładam, że większość Czytelników tego artykułu ma przede wszystkim doświadczenie obiektowe, z językami mocno typowanymi). Groovy powstał jako konglomerat różnych pomysłów, które na przestrzeni czasów pojawiły się w wartościowych językach programowania. Może przez to połączenie jest czasem nieco *przesadzony*, jednak jego poznanie może znacząco ułatwić podróże w kierunku języków odległych Javie. Siłą Grooviego (dla programistów Javy) jest to, że ucząc się nowej koncepcji, wciąż pozostajemy w świecie dobrze nam znanym i używamy tych samych bibliotek. Nie będzie to zatem nauka od zera.

Artykuł dedykowany jest dla osób, które o Groovim w ogóle nic nie wiedzą albo jedynie coś, gdzieś, kiedyś słyszeli. Jeśli masz już, drogi Czytelniku, pewne doświadczenie z Groovim i interesujesz się tworzeniem aplikacji internetowych, zachęcam do przeczytania artykułu Mariusza Roga pt. *Grails, Groovy i rusztowania* na temat szkieletu Grails. Artykuł opublikowany jest w niniejszym wydaniu SDJ.

Java script (ale nie JavaScript)

Kiedy powstał język Java, zachwycił on wielu programistów możliwością bardzo szybkiego wytwarzania aplikacji (zwłaszcza sieciowych lub bazodanowych) w porównaniu do C++. Narzucona na poziomie koncepcji języka konieczność zachowywania pewnego porządku w kodzie, czytelność i łatwość nauki spowodowały, że wybaczono Javie ułomność związaną z jej ówczesną wydajnością i zaczęto się nią poważnie interesować. Czy jednak kodowanie w Javie jest naprawdę takie szybkie? Wiele jej mechanizmów i cech (np. mocna kontrola typów) zostało stworzonych po to, by ustrzec nas przed popełnianiem błędów, zwłaszcza tych niewygodnych do wykrycia. Dzięki takiej koncepcji osiągnęliśmy język, którego kod jesteśmy w stanie łatwo weryfikować z użyciem narzędzi. Tym samym zarządzanie wielkimi projektami, nad którymi pracują tabuny programistów, jest zwyczajnie możliwe, a przynajmniej – prostsze. Czy jednak składnia języka powinna brać pod uwagę ludzką ułomność, narzucać coś, by człowiek nie popełnił błędu? Czy nie tworzymy tym samym ideologii *nie myśl – programuj*? Oczywiście celowo ubarwiam tę kwestię, zmierzam jednak tym samym do mniej kontrowersyjnego stwierdzenia, że języki dynamiczne, stojące niejako w opozycji do Javy, dają programiście dużo większą swobodę wyrażania myśli. Uważam, że dobrze się dzieje, iż tego typu rozwiązania zyskują coraz bardziej na popularności. Często języki dynamicznie typizowane, o skryptowej naturze, pozwalają zakodować coś szybciej, wyrazić myśl w bardziej naturalny sposób, a szablonowe fragmenty kodu po prostu w locie wygenerować. Rygor utrzymywania jakości kodu pozostaje w tym wypadku po stronie programistów, co może być bolączką podczas pracy zespo-

łowej. Można by zatem rzec, iż skrypty nie są dla idiotów. Chaos, jaki można stworzyć skryptem choćby w średniej wielkości projekcie, może pozostać nie do ogarnięcia nawet dla programistycznego guru. Czy to jednak powinno bronić nas przed używaniem mniej schematycznych (od Javy) języków programowania? Jedynie o Lispie i SQLu mówi się, iż programowanie w tych językach składa się przede wszystkim z myślenia, a zapisywanie kodu to jedynie trzydzieści procent czasu. Myślenie jednak nie boli, natomiast patrzenie na kod, który z łatwością można by wygenerować lub stworzyć z użyciem techniki kopiuj-wklej, może co wrażliwszych programistów doprowadzić do zapalenia spojówek.

Wystarczy już krytyki na temat Javy – język ten ma wiele dobrych cech. Przez lata Java dorobiła się ogromnej ilości bibliotek, szkieletów, narzędzi programistycznych – stabilnych i pisanych na profesjonalnym poziomie, często przez uznane instytucje. Stworzył się standard. Java jest jedną z najczęściej rozważanych alternatyw przy tworzeniu wielkich aplikacji korporacyjnych, integrujących różne środowiska, trzon dla wielu rozwiązań webowych. Żaden z języków wyższego poziomu nie otrzymał tak profesjonalnego wsparcia i pewnie żaden nie wygenerował twórcom programów takich zysków, jak Java. Przykładowo Python, który jest językiem przeestetycznym i wygodnym w pisaniu, boryka się z nadmiarem niejednorodnych i niekompletnych rozwiązań i bibliotek, często pisanych amatorsko, na kolanie. Nawet jeśli na dobrym poziomie, to bez niezbędnego wsparcia, zapewniającego profesjonalny rozwój rozwiązań. Kolejny język nadający się do szybkiego pisania aplikacji i bardzo naturalnego sposobu wyrażania myśli – JavaScript – praktycznie nie istnieje poza środowiskiem przeglądarek. Powstałe rozwiązania są wykorzystywane co najwyżej przez pasjonatów niż twórców poważnych aplikacji, a sam język, ze względu na marginalną wiedzę na jego temat pośród większości programistów, często nie cieszy się dobrą opinią.

W takiej sytuacji niesamowitym kompromisem byłby język, który pozwoliłby na skorzystanie z ogromu rozwiązań powstałych dla języka Java, jednak z drugiej strony miałby naturę bardziej dynamiczną, pozostawiającą programiście dużą dowolność. Taki język doskonale nadawałby się do pisania mikroprogramów lub małych narzędzi, jak i wielkich aplikacji. Taki język powinien, w razie konieczności, pozwalać na implementację części funkcjonalności bezpośrednio w języku Java lub w aplikacjach Java być wykorzystywanym do implementacji tych fragmentów, które łatwiej jest opisać skryptowo. Taki język.... to Groovy. Gdyby nazwa JavaScript nie została już (zupełnie nieadekwatnie) zarezerwowana, Groovy byłby najpewniej określany w ten właśnie sposób. Choć może to stwierdzenie nieco na wyrost – Groovy jest w końcu językiem kompilowanym, zachowuje jednak wiele cech języków skryptowych.

Trochę faktów

Oficjalny manual Grooviego we wstępie wita na pokładzie nowego języka, każe zapiąć pasy i wznieść się na wyżyny programowana. Lotnicze skojarzenie ma zapewne źródło w historii (a może – legendzie) powstania języka. Jego twórca – James Strachan – wpadł na pierwotną ideę Grooviego, przeglądając na lotnisku, w oczekiwaniu na samolot, artykuły na temat Pythona. Jako zagorzały programista Javy, nie miał wielkiego doświadczenia z językami bardziej dynamicznymi i doznał prawdziwego olśnienia, widząc, jak wielkimi skrótami myślowymi można posługiwać się, programując w Pythonie. Zaczął rozwijać koncepcję języka, czerpiąc z istniejących rozwiązań, starając się jednak zachować pełną zgodność z Javą. Podstawową ideą było utworzenie języka, który bez żadnych dodatkowych mechanizmów będzie mógł korzystać z ogromu bibliotek Javy, jak również, który z poziomu języka Java będzie rozpoznawany jako spójny fragment programu. Minęło prawie sześć lat od tego czasu. Język Groovy, dzięki zapałowi twórcy i jemu podobnych pasjonatów – powstał i znalazł swoją pozycję. Ma coraz większe grono zwolenników. Z drugiej strony, po sześciu latach od powstania, wielu programistów Javy nie zna nawet podstawowych koncepcji Grooviego, którego używanie nie musi być wcale *zdradą* Javy, a jedynie jej praktycznym uzupełnieniem. Właśnie im dedykowany jest ten artykuł.

Groovy jako koncepcja języka powstał w 2003 roku, jednak na pierwszą oficjalną i stabilną implementację (wersja 1.0) trzeba było czekać aż do 2007 roku. Jeszcze w tym samym roku została wydana znacznie poprawiona wersja 1.1, która dla podkreślenia zmian została szybko przemianowana na 1.5. W chwili pisania artykułu dostępna była wersja 1.6.3 z maja 2009 roku. Ostatnie dwa lata to okres silnego zainteresowania językiem oraz rozwoju dedykowanych szkieletów i narzędzi. Aktualnie trwają prace nad jego standaryzacją zgodnie z *Java Community Process* (JSR241).

Język Groovy definiuje się jako dynamiczny (często także pada określenie – skryptowy) język obiektowy dedykowany platformie Java. Najczęściej porównuje się go do takich języków jak Python, Ruby, Perl i Smalltalk. Groovy charakteryzuje się zarówno dynamiczną, jak i silną typizacją. Mimo że jest językiem kompilowanym, umożliwia znaczące modyfikowanie prototypów klas oraz dynamiczne uruchamianie kodu. Do podstawowych różnic w stosunku do języka Java należą: domknięcia, możliwość przeładowywania operatorów, wyrażenia regularne wbudowane w składnię języka, koncepcja właściwości klas (ang. *properties*) oraz możliwość ewaluacji wyrażeń.

Web is dead

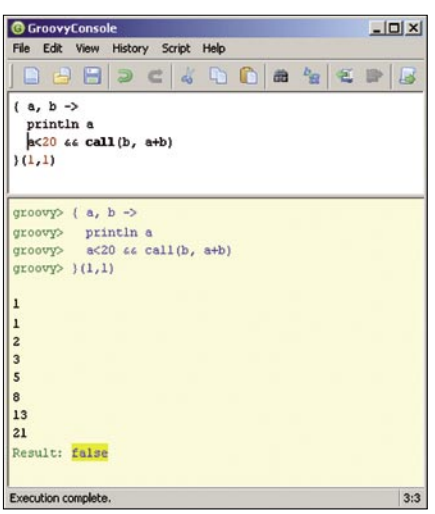
Web is dead, czyli nieco bardziej cyberpunkowy *Hello world*. W końcu i tak chodzi tylko o wygenerowanie jakiegokolwiek tekstu na ekranie. A ponieważ jest to offline'owy ekran konsoli systemowej, tekst będzie na miejscu. Niniejszym ogłaszam najkrótszy w historii SDJ rozdział poświęcony temu zagadnieniu. Oto rozwiązanie

```
groovy -e "print 'Web is dead'"
```

Ktoś chciał coś powiedzieć o statycznej metodzie `main` jakiejś tam klasy, przyjmującej w parametrze tablicę? Hmm...

Środowisko języka Groovy

Przedstawiony powyżej pierwszy program niewiele nam jeszcze mówi o samym języku i możliwości uruchamiania jego programów. Zajmijmy się najpierw tym drugiem. Groovy pozwala kompilować kod do plików *.class*, pozostając nieodróżnialnym od Javy dla maszyny wirtualnej tego języka. Może też, tak jak zrobiliśmy to w przypadku pierwszej aplikacji, uruchomić kod bezpośrednio z wiersza poleceń – dzięki przełącznikowi `-e` polecenia `groovy`. Kolejną, bardzo wygodną w trakcie nauki opcją jest konsola (uruchamiana poleceniem `groovyConsole`), w której szybko możemy pisać małe fragmenty kodu i obserwować ich działanie.



Rysunek 1. *Konsola języka Groovy pozwala w łatwy sposób testować małe programy, bez konieczności jawnej kompilacji*

Wygląd konsoli języka zaprezentowano na Rysunku 1.

Pracując z konsolą, bądź uruchamiając programy z użyciem narzędzia *groovy*, możemy odnieść wrażenie, że Groovy jest językiem interpretowanym. Gdyby tak było, jego integracja z Javą nie byłaby już taka prosta. W rzeczywistości Groovy zawsze tworzy pliki *class*, choć czasem może to być ukryte dla programisty – kompilator może umieszczać klasy w plikach tymczasowych lub bezpośrednio w pamięci. Zazwyczaj jednak będziemy jawnie odwoływać się do standardowych plików *.class*. By je otrzymać – potrzebujemy kompilatora, a więc narzędzia `groovyc`. Polecenie

```
groovyc *.groovy
```

utworzy pliki *.class* dla zadanego katalogu. Od tego momentu będziemy mogli skorzystać z nich w jakimś projekcie języka Java lub zwyczajnie uruchomić. Ciekawą opcją kompilatora *groovyc* jest możliwość użycia narzędzia *javac* celem jednoczesnego skompilowania plików *.groovy* i *.java* i rozwiązania zależności tkwiących między tymi plikami. Służy do tego przełącznik `-j`:

```
groovyc -j Test.groovy Main.java
```

Skompilowane pliki *.groovy* będziemy uruchamiać tak jak każdy program Javy. Musimy tylko pamiętać o dodaniu do ścieżki niezbędnych bibliotek. Ponieważ ich ilość jest niemała, możemy skorzystać z dystrybucji języka, w której wszystkie niezbędne pliki są osadzone w jednym pliku *jar*, zajmującym ponad 4MB. Uruchomienie skompilowanej klasy Groovy (*Test.class*) będzie zatem wyglądało następująco:

```
java -cp groovy-all-1.6.0.jar; Test
```

Podstawowa składnia

Wiemy już, jak uruchamiać programy języka Groovy, skupmy się zatem na prostych przykładach mających na celu zobrazowanie cech języka. Zachęcam do uruchamiania ich z poziomu konsoli *groovyConsole*. Zakładam, że Czytelnicy tego artykułu znają przynajmniej na podstawowym poziomie Javę, postaram się więc położyć szczególny nacisk na wskazanie różnic między tym językiem a Groovym.

Przedstawiony wcześniej pierwszy, jednowersowy przykład (*Web is dead*) wskazał już na kilka istotnych różnic (i skrótów) przyjętych przez język Groovy. Funkcja `print` została użyta bez nawiasów, zabrakło też końcowego średnika. Oba te elementy nie są wymagane, o ile jasno z kontekstu można zrozumieć intencje wykonania kodu. Średnik zwykle przyda się w przypadku umieszczenia w jednej linii programu kilku niezależnych wywołań. Pomijanie nawiasów w wykonaniu funkcji zwykle może zagmatwać czytelność, choć zwyczajowo opuszcza się je dla funkcji `print`. Ponadto będzie to bardzo przydatne przy pracy z domknięciami, o czym w dalszej części artykułu.

Podstawową różnicą między Javą a Groovim jest możliwość używania dynamicznej typizacji obiektów. Silna typizacja, będąca cechą języka Java, niejednokrotnie wymusza konieczność częstego jawnego rzutowania. Groovy rozwiązuje ten problem w sytuacjach, w których nie jest to potrzebne, w innych pozwala natomiast na zachowanie kontroli typów. Oto przykład:

```
def a=1
Integer b = 1
a = "test"
b = "test" //błąd
```

W języku Groovy nie musimy tworzyć klasy, by napisać prosty kod (kompilator utworzy ją w takiej sytuacji domyślnie). Możemy zatem bezpośrednio zająć się pisaniem kodu lub zamiast klas korzystać wyłącznie z funkcji. Deklaruje się je następująco:

```
def poleKola(promien) {
  Math.PI * promien**2
}
```

Powyższy kod ma kilka ciekawych cech. Po pierwsze – nie określamy typu zwracanego, wiemy już zatem, że będzie on miał dynamiczny charakter. Po drugie – odwołujemy się tu do właściwości `PI` klasy `Math` – tej samej, której używa język Java. Po trzecie – użyliśmy nieobecnego w Javie operatora potęgowania (`**`). Finalnie zaś – zwracamy wartość funkcji bez użycia słowa kluczowego `return`. Użycie go jest opcjonalne – w przypadku jego pominięcia zostanie wzięta pod uwagę wartość ostatniej linii kodu funkcji.

Niestandardowe operatory

Poniżej znajduje się opis operatorów języka Groovy, które nie mają swoich bezpośrednich odpowiedników w Javie lub które w Javie występują, lecz zyskały tutaj dodatkowe znaczenie.

- `a ** b` – potęgowanie; podnosi `a` do potęgi `b`.
- `*lista` – konwertuje listę do ciągu oddzielnych parametrów; w praktyce pozwala na przekazanie funkcjom parametrów w postaci listy, np. `"text".substring(*[1,2])`; operator może też być użyteczny przy dodawaniu nowych elementów do listy lub mapy, np.: `[1,2,3,*[4,5,6]]` utworzy sześcioelementową, płaską listę.
- `a[b]` – udostępnia obiekt pod wskazanym indeksem, działa zarówno z tablicami, jak również listami i mapami; umożliwia odczyt i zapis danych.
- `a << b` – klasyczne przesunięcie bitowe w lewo, jednak dla niektórych typów posiada przeładowane znaczenie typu *dodaj, uzupełnij o element*. W tej formie operuje na obiektach `StringBuffer`, klasach typu `Writer`, plikach, gniazdach, listach i metaobiektach.
- `<=>` – komparator, ze względu na wygląd nazywany statkiem kosmicznym (ang *spaceship*); służy do porównywania wartości obiektów.
- `a as typ` – tworzy wymuszenie rzutowania bądź alias dla jakiegoś typu. W tym pierwszym przypadku możemy użyć operatora np. do zainicjalizowania obiektu z użyciem listy: `def obj = [1,2,3] as MyClass`. Aliasowanie pozwala natomiast na uniknięcie konfliktów nazw, poprzez ich import pod nową nazwą, np. `import java.sql.Date as SQLDate`
- `a .@ b` – bezpośredni dostęp (z pominięciem metody `get`) do składnika `b` obiektu `a`.
- `a .& b` – zwraca referencję do metody `b` obiektu `a` w postaci domknięcia; kontekst obiektu zostaje zachowany (odwołania z użyciem `this`); operator może mieć zastosowanie np. przy przekazywaniu metody w postaci parametru do funkcji oczekujących na wejściu domknięcia (np. `lista.each(obiekt.&metoda)`)
- `a .. b` – operator zakresu, np `(1..20)`; istnieje możliwość stworzenia przedziału prawostronnie otwartego, np. `(100..<105)`.
- `a *. b` – odnosi się do właściwości lub metody `b` każdego obiektu zawartego w kolekcji `a`; przykładowo kod `["a","b","c"]*.toUpperCase()` zwróci w rezultacie listę `["A","B","C"]`.
- `a ?. b` – opcjonalny dostęp do elementu; `b` zostanie użyte, tylko jeśli `a` nie ma wartości `null`; pozwala uniknąć pisania dodatkowych warunków i znacząco zmniejsza ryzyko wystąpienia w kodzie wyjątków typu *Null pointer exception*.
- `==~` – operator dopasowania w wyrażeniach regularnych.
- `=~` – operator przeszukiwania w wyrażeniach regularnych.

GString

Operacje na łańcuchach znaków są często niezwykle istotnym elementem programów, a przy niewygodnych mechanizmach potrafią znacząco utrudnić programistom życie. Groovy wychodzi tu na pomoc z rozbudowaną koncepcją obsługi tekstów i wbudowanym w język wsparciem dla wyrażeń regularnych. Oprócz implementacji klasycznych obiektów `java.lang.String`, znajdziemy tu rozszerzenie w postaci klasy `groovy.lang.GString`, która umożliwia dostęp do zmiennych z poziomu łańcuchów znaków, jak ma to miejsce w wielu językach skryptowych:

```
public class Person {
  def firstname, lastname
  String toString() {
    "$firstname $lastname"
}}
```

Jak widzimy na tym prostym przykładzie, łatwo możemy pozbyć się niewygodnej konkatenacji łańcuchów. Może mieć to zastosowanie w wielu sytuacjach, np. w servletach, przy tworzeniu zapytań SQL itp.

Rozróżnienie między klasycznym *Stringiem* a *GStringiem* odbywa się po użytych znakach cytowania. Pojedynczy cudzysłów tworzy obiekty klasy `java.lang.String` i w tej sytuacji nie ma możliwości interpolacji zmiennych. Podwójne cudzysłowy umożliwiają skorzystanie z zalet obiektów `GString`.

Groovy wprowadza jeszcze jedno bardzo ciekawe udogodnienie, umożliwiające operowanie na długich, wielowersowych łańcuchach znaków. Służą do tego pojedyncze lub podwójne cudzysłowie (w zależności, czy chcemy rozwijać wartości wyrażeń, czy nie), użyte trzykrotnie:

```
def txt = """Dzisiaj jest ${new Date()}.
  Test długiego napisu w języku Groovy"""
```

Kolejnym rozszerzeniem jest możliwość używania operatorów << i <<= do konkatenacji stringów. Jest to dość czytelne odróżnienie operacji łączenia napisów od dodawania numerycznego (choć operatory + i += również funkcjonują prawidłowo na tekstach). Najbardziej przydatna jest tu możliwość użycia operatora << na obiektach klasy `StingBuffer` zamiast metody `append`.

Pisząc o operacjach na tekstach, nie sposób nie wspomnieć o wyrażeniach regularnych, które w Groovim mają pełne wsparcie bezpośrednio z poziomu języka. Przyjrzyjmy się zupełnym podstawom, skupiając się przede wszystkim na składni (wyrażenie regularne to osobny temat, wykraczający daleko poza zakres tego artykułu). Definicja wzorca wyrażenia regularnego odbywa się z użyciem dedykowanych operatorów, celem odróżnienia od zwykłych łańcuchów znaków:

```
def pattern = ~/a(.*)d/
```

Możemy teraz spróbować porównać dowolny napis ze zdefiniowanym wzorcem – służy do tego operator ==~:

```
if( 'abc' ==~ pattern ) { … }
```

Chcąc otrzymać wszystkie odpowiadające wzorcowi dopasowania, należy skorzystać z kolejnego operatora, który w rezultacie zwróci listę wyników:

```
def matches = 'abcd' =~ pattern
```

Dla naszego przykładu będzie to dwuelementowa lista `['abcd', 'bc']`

Tworzenie klas – podstawy

Definiowanie klas w języku Groovy w wersji podstawowej przypomina mocno Javę. Na wyższym poziomie szczegółowości pojawia się tu jednak wiele elementów rozbudowujących bazową składnię. W niniejszym opisie stworzymy, krok po kroku, przykładową klasę, skupiając się na istotnych cechach języka. Niech będzie to klasa opisująca języki programowania. Zacznijmy od kilku podstawowych pól:

```
public class ProgrammingLanguage {
  String name
  def author
  String helloWorldCode
}
```

Pierwszą dostrzegalną różnicą jest użycie dynamicznego typu (`def`). Dzięki temu polu `author` będziemy mogli, w zależności od potrzeby, przypisać np. prosty napis, obiekt klasy `Person` lub listę tych obiektów.

Nigdzie w przykładzie nie umieściliśmy operatora zakresu – w przypadku Grooviego nie oznacza to jednak, że składowe klasy przyjmą domyślnie zakres pakietu. Dla metod domyślnym zakresem jest `public`, natomiast dla pól – `private`.

Ponieważ jak dotąd nie zdefiniowaliśmy w klasie żadnego konstruktora, mamy, podobnie jak w Javie, do dyspozycji bezparametrowy konstruktor domyślny. Ponadto standardowo Groovy udostępnia jeszcze jeden konstruktor, dzięki któremu możemy w trakcie inicjalizacji obiektu ustawić wartości pól za pomocą mapy:

```
def pl = new ProgrammingLanguage(name:
   "Perl", author: "Larry Wall")
```

Konstruktory te będą dostępne dopóty nie zdefiniujemy własnych. Implementacja konstruktora wygląda tak samo jak w Javie:

```
ProgrammingLanguage(String name) {
  this.name = name
}
```

Gdybyśmy konstruowali naszą przykładową klasę w Javie, z pewnością uzupełnilibyśmy ją o metody *get* i *set* dla poszczególnych pól, zwłaszcza że są one prywatne i nie powinniśmy mieć do nich bezpo-

Listing 1. *Przykład użycia budowniczego SwingBuilder do szybkiego tworzenia interfejsu użytkownika*

```
import groovy.swing.SwingBuilder

swing = new SwingBuilder()
frame = swing.frame(title: 'Groovy Builder') {
  menuBar {
    menu('Help') {
      menuItem 'Help'
      menuItem 'About'
    }
  }
  panel {
    boxLayout(axis: javax.swing.BoxLayout.Y_AXIS)
    label 'Hello World'
    button (
      text: 'Exit',
      actionPerformed: { event -> System.exit(0)}
    )
  }
}

frame.pack()
frame.show()
```

średniego dostępu. Zróbmy eksperyment w Groovim:

```
def sch = new ProgrammingLanguage("Scheme")
sch.author = ["Guy L. Steele", "Gerald Jay
                    Sussman"]
```

Zadziwiająco – powyższy kod skompiluje się. Co więcej, również taki kod nie zwróci żadnego błędu:

```
sch.setHelloWorldCode('(display "Hello
                    World!")')
```

Myślę, że osoby znające np. podstawy języka C# mogą już wiedzieć, co się stało. Groovy chroni nas przed koniecznością pisania oczywistego kodu, którego apogeum stanowią wszechobecne metody *get* i *set*. Zaledwie w znikomej ilości przypadków metody te robią cokolwiek ponad ustawianie wartości pól prywatnych. Łatwo można je zatem generować, co jest cechą wielu środowisk programistycznych języka Java. W przypadku Grooviego – zrobi to za nas sam język! Uporządkujmy wiedzę – deklarując pole klasy bez podania zakresu widoczności, tworzymy de facto właściwość (ang. *property*), która jest prywatna, automatycznie jednak zostaną dla niej wygenerowane publiczne metody *get* i *set*. Możemy odwoływać się do nich przez nazwę, jednak również bezpośrednie odwołanie do pola klasy uruchomi w rzeczywistości metodę. Jedynie w obrębie samej klasy będziemy korzystać bezpośrednio ze zmiennych, a nie metod. Stwórzmy na przykład metodę `toString`:

```
String toString() {
   "$name, created by ${getAuthor()}"
}
```

Ponieważ jest to metoda tej samej klasy, nie mamy potrzeby korzystania z metody `get`. W razie konieczności możemy jej użyć, jednak należy to zrobić jawnie, tak jak pokazano to na przykładzie dla właściwości `author`.

Gdybyśmy w ogóle nie chcieli mieć metod dla jakiegoś pola, wystarczy zadeklarować je jako `private`. Groovy generuje metody tylko dla właściwości o niezdefiniowanym zakresie. Nic też nie stoi na przeszkodzie, by nadpisać metodę pobierającą wartość lub ją zwracającą. Będziemy musieli to zrobić dla sytuacji, w których metoda będzie miała za zadanie wykonać dodatkowe czynności lub kiedy będziemy chcieli ograniczyć jej widoczność (np. do `protected`). Jeśli właściwość klasy zostanie zadeklarowana jako `final`, zostanie naturalnie utworzona jedynie metoda `get`.

Wiemy już, jak definiować klasy, ich pola, metody i specjalne pola – właściwości. Podobnie jak w przypadku języka Java, klasy mogą dziedziczyć po sobie z użyciem konstrukcji `extends NazwaNadklasy`. Natomiast mamy tu ograniczenia w postaci braku możliwości używania klas wewnętrznych (możemy zamiast nich skorzystać z domknięć). Również interfejsy wymagają specjalnego traktowania. Zostało to opisane w dalszej części artykułu.

Tablice, listy, mapy i zakresy

W Groovim znacznie częściej niż w Javie będziemy operować na tablicach i mapach dzięki możliwości ich szybkiego, dynamicznego tworzenia. Tego typu konstrukcje są zresztą domeną wielu języków dynamicznych – w Lispie praktycznie wszystko jest listą, w JavaScripcie każdy obiekt będzie mapą (tablicą asocjacyjną) itp.

Groovy umożliwia tworzenie kolekcji w sposób identyczny jak w Javie, jednak częściej korzystać będziemy ze znacznie szybszej notacji operatorowej:

```
def lista = [1,2,3,"tekst", new Object()]
```

Zapis powyżej tworzy kolekcję typu `ArrayList` i wypełnia ją pięcioma obiektami. Używamy tu dynamicznej typizacji, zatem obiekty umieszczane na liście mogą być dowolnego typu.

W sposób zbliżony do list tworzyć będziemy mapy (a konkretnie obiekty klasy `LinkedHashMap`):

```
def colors = [ white: "#FFFFFF", red:
                    "#FF0000" ];
```

Korzystając z powyżej notacji, łatwo możemy utworzyć pustą listę (`[]`) oraz pustą mapę (`[:]`).

Definicja tablicy (jest to raczej rzadko używana w Groovim struktura danych) wymaga określenia konkretnego typu tablicowego (celem odróżnienia od listy), np.

```
int[] numbers = [1,2,3]
```

Zwróćmy uwagę, że elementy tablicy są zainicjalizowane tak jak lista – z użyciem kwadratowych nawiasów. W Javie użylibyśmy tu nawiasów klamrowych.

Specyficznym typem, który nie posiada swojego bezpośredniego odpowiednika w Javie, jest zakres (ang. *range*) definiujący ciąg kolejnych elementów:

```
def zakr = 100..200
```

Zakresy mogą mieć wiele praktycznych zastosowań, jak np. iterowanie pętli po kolejnych elementach:

```
for( i in 1..10 ) { ... }
```

lub wypełnianie list kolejnymi wartościami (przykład tworzy listę elementów od 1 do 5 oraz od 20 do 25, czyli łączy ze sobą dwa zakresy):

```
list = [*(1..5), *(20..25)]
```

Operatory i ich przeładowywanie

Język Groovy wprowadza kilka ciekawych operatorów. Część z nich opiszemy bar-

Iterowanie z Groovim

Groovy rozszerza klasę `java.lang.Object` o dodatkowe metody ułatwiające iterowanie po elementach z użyciem domknięć, co praktycznie zwalnia programistę z konieczności używania pętli. Metody te można stosować dla wszelkiego rodzaju kolekcji, enumeracji, zakresów i łańcuchów znaków. Poniżej przedstawiono ich listę wraz z przykładami użycia. Każda z opisanych tu metod przyjmuje w parametrze domknięcie.

- `any` – zwraca wartość `true`, jeśli domknięcie zwróci taką wartość dla przynajmniej jednego z elementów. Przykładowo, kod `[null, null, 1, null].any{ it != null }` sprawdzi, czy lista zawiera niepuste elementy.
- `collect` – zwraca listę wartość zwróconych przez domknięcie dla każdego z elementów wejściowych. Przykładowo, `"text".collect{ it.charAt(0) }` zwróci listę znaków składających się na napis wejściowy.
- `each` – uruchamia przekazane domknięcie dla każdego elementu listy wejściowej. W rezultacie zwraca listę wejściową.
- `eachWithIndex` – podobnie jak `each`, ale domknięcie przyjmuje jeszcze drugi parametr, będący indeksem obiektu w kolekcji.
- `every` – działa podobnie jak `any`, jednak zwraca wartość `true`, tylko jeśli dla wszystkich elementów wejściowych domknięcie zwróciło wartość `true`.
- `find` – przekaże w rezultacie pierwszy element z listy, dla którego domknięcie zwróci wartość `true`.
- `findAll` – jak wyżej, ale wygeneruje listę wszystkich wartości zgodnych z domknięciem wejściowym. Przykładowo, kod `(1..50).findAll{ it % 2 == 0 }` zwróci listę liczb parzystych w zadanym zakresie.
- `findIndexOf` – podobnie jak `find`, ale zwraca indeks zgodnego elementu.

dziej szczegółowo w dalszej części tekstu, natomiast wszystkie niestandardowe operatory (w stosunku do Javy) przedstawiono w osobnej ramce w postaci szybkiej ściągi. Nowe operatory mają zwykle na celu bardziej skrótowe zapisywanie często powtarzanych operacji. Przykładowo, dobrą praktyką w języku Java jest sprawdzanie, czy obiekt, na którym operujemy (zwłaszcza kiedy otrzymaliśmy go np. w postaci parametru metody), nie ma czasem wartości `null`. W efekcie powoduje to pisanie dużej ilości operacji warunkowych. Groovy pozwala ten nadmiar kodowania uprościć z użyciem operatora `?.`, który umożliwia dostęp do znajdującego się po kropce elementu, tylko jeśli obiekt jest zdefiniowany:

```
String s = null
print s?.toUperCase?.charAt(1)
```

Jak widzimy, możemy tego typu operatory łączyć w obrębie jednego ciągu wyrażeń. Jeśli na którymkolwiek etapie otrzymamy w rezultacie wartość `null`, operacja zostanie przerwana bez zwracania błędu.

Kolejnym operatorem, który może mieć częste zastosowanie, jest `*.`. Jest to szybki dostęp do pól lub metod wszystkich elementów kolekcji. Rezultat jest zwracany w postaci listy. Dzięki temu prostemu operatorowi możemy zaoszczędzić na pisaniu pętli przechodzących przez wszystkie elementy kolekcji. Zobaczmy przykład:

```
[10, 3.14, "text"]*.toString()*.reverse()
```

Efektem działania powyższego kodu będzie lista następujących wartości: `["01", "41.3", "txet"]`.

W językach obiektowych użycie operatora często jest jedynie zapisem skrótowym dla metod implementujących określoną operację. Tak jest również w języku Groovy. Co więcej, programista ma tutaj możliwość przeładowania operatorów, definiując specyficzne zachowania dla własnych klas. Każda z funkcji operatorowych ma opisową nazwę, wskazującą na operator, np. `plus`, `minus`, `power`, `rightShift` etc. W przypadku operatorów dwuargumentowych, lewą stronę reprezentować będzie obiekt klasy, dla której implementujemy funkcję operatorową, a prawą stronę – parametr metody. Operator sumowania obiektów hipotetycznej klasy `Money` z wartościami liczbowymi mógłby zatem wyglądać następująco (`plus` jest metodą klasy `Money`):

```
Money plus(int val) {
  new Money(this.amount+val, this.currency)
}
```

W opisany tu sposób możemy przeładowywać prawie wszystkie operatory, włączając te specyficzne dla języka Groovy. Wyjątkiem są `?:`, `?.`, `*.`, `.`, oraz jednoargumentowe operatory `*` i `&`. Ciekawostką natomiast może być fakt, że wybór alternatyw z użyciem słów `switch`/`case` jest zdefiniowany w Groovym jako operator, który dodatkowo można przeładować (z użyciem metody `isCase`). Więcej o samym operatorze w kolejnym punkcie.

Instrukcje sterujące

Podstawowe instrukcje sterujące będą zauważalnie przypominały te znane z języka Java (`if`, `while`, `for`), bez znaczących różnic. Jak wszędzie w Groovym, wprowadzono tu zapisy skrótowe, wynikające choćby z nierestrykcyjnej kontroli typów. Przykładowo, możemy sprawdzać zainicjalizowanie obiektów bez konieczności porównywania ich do `null`, bowiem wartość ta jest w porównaniach równa logicznej wartości `false`:

```
Object obj = null
if( obj ) { ... }
```

Pamiętajmy jednak, że również wartość zero i pusty napis zwracają w porównaniach wartość `false`.

Pętla `for`, podobnie jak w nowszych wersjach języka Java, występuje w dwóch wersjach, przy czym wersja `for...in` ma tutaj dużo większe możliwości. Groovy pozwala z użyciem tej instrukcji iterować praktycznie po wszystkich obiektach, które składają się z listy składowych elementów. Będą to zatem tablice, enumeracje, mapy, zakresy, a nawet łańcuchy znaków (w tym wypadku każdy znak zostanie potraktowany jako kolejny element iteracji). Poniżej kilka przykładów:

```
for( x in 1..10) { println x }
for( x in "abcdef" ) { println x }
for( x in [ 1:"a", 2:"b", 3:"c" ] ) {
                        println x.key }
```

Osobnego komentarza wymaga instrukcja `switch`, która w Groovim została znacząco rozbudowana, pozostawiając jednak kompatybilność wsteczną. Porównywanymi wartościami (znajdującymi się przy słowie kluczowym `case`) mogą tu być dowolne obiekty, włączając napisy, tablice, zakresy, mapy itp., np:

```
switch( v ) {
  case "text": //..
  case 1..100: //..
  case [1,2,3,4]: //..
  default: //..
}
```

Listing 2. *Przykład użycia obiektu Eval, dynamicznego generowania metod oraz zastosowania domknięć. Program jest kalkulatorem pozwalającym rejestrować (w postaci nowych metod klasy) dodatkowe, nazwane funkcje*

```
public class DynCalc {
  public static calculate(String expression) {
    Eval.me(expression)
 }

  public static addFunction(name, fun, nr) {
    String evs = (["me", "x", "xy", "xyz"])[nr]
    String pas = (["", "a", "a,b", "a,b,c"])[nr]
    Closure ee = { expr, params -> Eval."${evs}"(*params, expr) }
    String clo = "{ ${pas} -> x(y,[${pas}]) }"
    DynCalc.metaClass."${name}" = Eval.xy(ee,fun,clo)
  }
}

DynCalc.addFunction2("G", "6.67428", 0)
DynCalc.addFunction("distance","x*y+z*y**2*0.5", 3)
DynCalc.addFunction("parabolic",'''
    def g=9.80665;
    [x*y*Math.cos(z), x*y*Math.sin(z)-0.5*g*y**2]
  ''', 3);

d=new DynCalc();
println d.calculate( "1+8*2/33" )
println d.G()
println d.distance(10,60,0.5)
println d.parabolic(3,30,45)
```

Szczególnym przypadkiem jest możliwość umieszczenia w warunkach nazw klas, co może nam pomóc zidentyfikować typ zmiennej:

```
switch(1) {
  case Number: print "to jesli liczba";
                     break
  case String: print "to jest napis";
                     break
  }
```

Taka wersja operacji `switch` dokonuje porównań bezpośrednich, nie powinniśmy używać jej zatem do sprawdzenia rzeczywistego typu zmiennej polimorficznej.

Pisząc o instrukcjach sterujących, należy jeszcze wspomnieć o możliwości obsługi sytuacji wyjątkowych. Groovy, w identyczny sposób jak Java, wprowadza bloki kodu typu `try – catch – finally`. Możemy rzucić wyjątek z użyciem słowa kluczowego `throw` oraz wskazać w deklaracji metody na potencjalnie rzucane przez nią wyjątki. Podstawowa różnica polega na braku konieczności ich przechwytywania. Wywołanie metody zwracającej wyjątek z pomięciem bloku `try` jest w Groovim dozwolone. To programista decyduje, jakie błędy i kiedy zamierza przechwytywać.

Instrukcja assert

Od wersji 1.4 języka Java dysponujemy instrukcją `assert`, która, zazwyczaj w fazie tworzenia oprogramowania, pozwala zweryfikować poprawność wartości wyrażeń w uruchamianym kodzie. Mechanizm ten jest szeroko stosowany w Groovim i prawie nie różni się składniowo od tego użytego w Javie. Główną różnicę znajdziemy natomiast w strategii używania. W języku Groovy nie dysponujemy możliwością wyłączenia asercji, co oznacza, że weryfikacja będzie odbywać się również w aplikacjach produkcyjnych. Koncepcja ta ma pewnie tyle samo wrogów, co zwolenników.

Instrukcja `assert` w programach Grooviego jest regularnie wykorzystywana i spotkamy się z nią w większości programów przykładowych oraz w różnego rodzaju dokumentach poświęconych językowi (począwszy od oficjalnej dokumentacji języka). Autorzy Grooviego postanowili używać w przykładach asercji zamiast funkcji `print` – ta ostatnia wypisze co prawda wynik działania programu na ekran, ale przy bardziej złożonych wyrażeniach nie będziemy umieli sobie tak naprawdę uzmysłowić wyniku bez uruchomienia programu (autorzy aplikacji przykładowych w takich sytuacjach często podają oczekiwany wynik w komentarzach). Dzięki zastosowaniu instrukcji `assert`, Czytelnik od razu widzi efekt działania programu. Zobaczmy przykład:

```
def a = 5
assert a << 5 == 20
```

Nie każdy programista potrafi wyliczyć w głowie wynik przesunięcia bitowego. Instrukcja `assert` od razu wskazuje na oczekiwany wynik. Prawidłowym rezultatem tak uruchomionego programu będzie zatem brak jakiegokolwiek wyniku na ekranie. Jednak gdybyśmy popełnili gdzieś błąd, otrzymalibyśmy komunikat *assertion error* wskazujący na konkretną instrukcję. Gdybyśmy sami chcieli zdefiniować treść wypisanego w takiej sytuacji komunikatu, należy skorzystać z drugiego parametru instrukcji `assert` (w odróżnieniu od Javy, podaje się go po przecinku, a nie po dwukropku):

```
assert 5**5==26,  "błędny wynik potęgowania"
```

Domknięcia (dla niewtajemniczonych)

Kto nie wie, czym są domknięcia (ang. *closures*), ten zapewne przespał zamknięty (domknięty?) w świecie Javy kilka lat (co nie znaczy, że przespał powstanie koncepcji domknięć, która zrodziła się już w 1960 roku). Domknięcia, wywodzące się pierwotnie z języków funkcyjnych, stają się coraz bardziej doceniane pośród programistów obiektowych i coraz więcej współczesnych języków zostaje uzupełnianych o ten mechanizm. Koncepcja ta pojawiła się nawet w planowanym nowym standardzie języka C++ (C++0x) – miejmy nadzieję, że kiedyś zawita także do Javy (wciąż trwa nierozwiązana debata, czy domknięcie powinny znaleźć się w Javie 7).

Czas najwyższy wyjaśnić, czym są te tajemnicze konstrukcje, najlepiej z pominięciem zbytniej teorii i skupieniu się na języku Groovy (mam nadzieję, że puryści wybaczą mi przyjęte uproszczenia). Najprościej rzecz ujmując – domknięcia to fragmenty kodu (nazwane lub anonimowe), które zachowują się jak funkcje (mogą mieć parametry wejściowe i zwracać wartości), jednak też posiadają pewne cechy obiektów – możemy je traktować tak jak zmienne – przekazywać jako parametry itp. Tyle teorii, bez przykładu się nie obędzie:

```
square = { it * it }
```

Utworzyliśmy domknięcie, które potrafi liczyć kwadrat zadanej liczby. Definicja powyżej wygląda jak deklaracja zmiennej, jednak użycie domknięcia będzie przypominało funkcję:

```
x = square(5)
```

Jak się można domyślić – w rezultacie otrzymamy liczbę 25. `it` w definicji domknięcia to domyślna nazwa parametru wejściowego (jeśli trzeba, możemy zdefiniować własny). Wartość wynikowa, tak jak w każdej metodzie języka Groovy, jest domyślnie brana z ostatniej linii kodu, można jednak wyraźnie określić wynik poprzez słowo kluczowe `return`. Moglibyśmy zatem zapisać to samo domknięcie w bardziej przegadany sposób (tylko po co?):

```
square = { num -> return num*num }
```

Skoro domknięcie tak bardzo przypomina funkcję, dlaczego w ogóle jest nam potrzebne? Wszystko zależy od sposobu stosowania. Mechanizm domknięć pozwala znacząco ułatwić życie za sprawą faktu, że możemy go traktować jako zmienną. Wyobraźmy sobie taką rzeczywistą funkcję:

```
def fun(liczba, operacja) {
  return operacja(liczba)
}
```

Przyjmuje ona na wejściu dowolną liczbę i dowolne domknięcie. To w zależności od niego funkcja będzie w stanie wykonać pewną operację na liczbie – taką, o jaką ją poprosimy:

```
fun( 5, square )
```

W przykładzie powyżej przekazaliśmy domknięcie tak, jak zmienną – bez użycia nawiasów. Gdybyśmy napisali `square()`, w rzeczywistości przekazalibyśmy wynik domknięcia, a to nie było naszym celem.

Mam nadzieję, że powoli dochodzimy do rozróżnienia między funkcją a domknięciem. Przydałby się jeszcze tylko bardziej obrazowy przykład, niż abstrakcyjna operacja na liczbach przedstawiona wyżej. Taki przykład weźmiemy ze standardowej biblioteki języka. Dla obiektów reprezentujących listę w Groovim mamy do dyspozycji metodę `collect`, która jako parametr pobiera domknięcie i wykonuje je dla każdego elementu listy.

Jednym słowem, chcąc policzyć kwadraty dla zadanego ciągu liczb, zrobimy to następująco:

```
[1,3,5,7].collect( square )
```

W rezultacie otrzymamy listę kwadratów liczb. Obyliśmy się bez pętli i mnó-

stwa zbędnego kodu! Parametrem metody `collect` jest domknięcie, możemy zatem dla każdego elementu listy wykonać dowolną operację, bowiem domknięcie może zawierać dowolny kod. Jedynym obciążeniem jest konieczność zadeklarowania samego domknięcia. Jednak.... wspominałem na początku, że domknięcia mogą być anonimowe. Co to znaczy? Wyjaśnijmy to na prostych zmiennych. Możemy napisać tak:

```
String a = "test"
print a
bądź tak
print "test"
```

W obu przypadkach utworzyliśmy obiekt typu `String`, jednak w tym drugim – anonimowo, nie przypisując go do żadnej zmiennej. Skoro domknięcia mają cechy zmiennych, możemy je zatem tworzyć również anonimowo (użyta tu metoda `each` po prostu uruchamia przekazany kod dla każdego elementu kolekcji):

```
[ "do", "re", "mi" ].each( { print it } )
```

Tym oto sposobem w naturalny sposób wyrażamy to, co chcemy zrobić dla każdego (`each`) elementu listy. Groovy pozwala ten zapis jeszcze bardziej uprościć. Jeśli parametrem jakiejś funkcji jest domknięcie i chcemy je w locie zaimplementować (zamiast przekazywać nazwę domknięcia), możemy opuścić okrągłe nawiasy. Groovy zrozumie, że w tym miejscu implementujemy anonimowe domknięcie:

```
[ "do", "re", "mi" ].each{
  it.reverse()
  print it
}
```

To już nie wygląda jak zwykłe wywołanie funkcji, to po prostu wygląda tak jakbyśmy funkcji `each`, która nie wie nic na temat tego, co ma zrobić z elementami listy, przekazali kawałek kodu do wykonania. Specjalnie rozbiłem go na kilka linii, by było widać, że domknięcie nie musi być jednoliniowe.

Bardzo ważną cechą domknięć jest fakt, że zapamiętują one wartości zmiennych, które w momencie ich tworzenia znajdowały się w ich zakresie widoczności. Brzmi skomplikowanie – zobaczmy przykład. Tym razem utworzymy funkcję, która zwraca w rezultacie domknięcie.

```
def createMultiplier(int val) {
  return { it * val }
}
```

Nasza funkcja tworzy nowe domknięcie, które będzie służyło do mnożenia liczb przez określoną wartość:

```
mulBy5 = createMultiplier(5)
print mulBy5(10)
```

Zwróćmy uwagę, że parametr `val` funkcji `createMultiplier` jest zmienną widoczną tylko w ciele funkcji. Ponieważ definicja domknięcia znajduje się wewnątrz funkcji, wszystkie jej zmienne będą dla domknięcia widoczne. Ważne jest, że nawet kiedy domknięcie znajdzie się poza ciałem funkcji (w tym wypadku zostanie przez nią zwrócone), wartości wewnętrznych zmiennych będą przez nie dalej pamiętane – oczywiście statycznie.

Powrócę jeszcze na chwilę do idei anonimowych domknięć. Wiemy już, że między domknięciem a jego uruchomieniem różnica leży jedynie w nawiasach. Możemy zatem natychmiastowo użyć domknięcia zwracanego przez funkcję, bez przypisywania go do zmiennej:

```
print createMultiplier(10)(100)
```

W niektórych sytuacjach czytelniej będzie zaznaczyć, że wymuszamy uruchomienie domknięcia. W tym celu Groovy definiuje metodę `call`, będącą składową wszystkich domknięć:

```
print createMultiplier(10).call(100)
```

W rzeczywistości notacja nawiasowa jest tylko skrótową formą wykonania metody `call`. Jeśli kogoś dziwi fakt, że domknięcie ma swoją metodę – przypominam, że fizycznie jest ono obiektem.

Sama koncepcja domknięć nie jest może zbyt skomplikowana, jednak pewnej praktyki wymaga, by zacząć myśleć i projektować kod z ich użyciem. Mogą się one okazać przydatne w wielu sytuacjach. Czasem tylko po to, by ładnie ograniczyć kod lub ustrzec się przed pisaniem prywatnych metod, które w całej klasie użyte są tylko raz. Wyobraźmy sobie, że mamy w metodzie jakąś zmienną lokalną, której inicjalizacja wymaga pewnych dodatkowych operacji. Dzięki domknięciom możemy napisać coś na kształt lokalnej anonimowej fabryki, która uratuje nas przed koniecznością zaśmiecania ciała metody dodatkowymi zmiennymi pomocniczymi (a tym bardziej – przed niepotrzebnymi metodami prywatnymi). Mogłoby to wyglądać na przykład tak:

```
long time = {
  try { new TimeService().getTime()
  } catch (Exception e) {
    System.currentTimeMillis()   }
}()
```

W przykładzie próbujemy zainicjalizować zmienną aktualnym czasem – o ile to możliwe, z użyciem jakiegoś zewnętrznego serwisu, jeśli jednak komunikacja z nim nie powiedzie się – z użyciem czasu systemowego. Jeśli tego typu funkcjonalność występowałaby jednokrotnie w obrębie programu, podane wyżej rozwiązanie, jako przypadek szczególny, będzie lepsze od tworzenia metody prywatnej. Zwróćmy uwagę, że w przykładzie tym utworzyliśmy anonimowe domknięcie, które natychmiastowo uruchomiliśmy (nawiasy na końcu).

Skoro już tyle powiedzieliśmy o anonimowości, warto wspomnieć o przypadku szczególnym – anonimowe domknięcie, które chcielibyśmy wykonać rekurencyjnie. Wiele języków funkcyjnych nie daje takiej możliwości. Znanymi mi wyjątkami są JavaScript (dzięki konstrukcji `arguments.calle`) oraz Groovy. W tym ostatnim sytuacja jest szczególnie prosta. Wspominałem wcześniej, że wykonanie domknięcia następuje dzięki składowej metodzie `call` – zatem wystarczy ją wykonać z wnętrza kodu. Kończąc rozważania o domknięciach (i zachęcając do eksperymentowania z nimi), zobaczmy przykład anonimowego rekurencyjnego domknięcia liczącego ciąg Fibonacciego (z pewnym ograniczeniem, by uniknąć przepełnienia stosu):

```
{ a, b ->
  println a
  a<20 && call(b, a+b)
}(1,1)
```

Wykorzystać ale nie poślubić

Żaden język programowania nie ma racji bytu bez dedykowanych dla niego bibliotek (no, chyba że SQL). W obecnych czasach tylko nieliczni zajmują się wymyślaniem klocków – reszta buduje z nich swe aplikacje. Język Groovy, *ex definitione*, klocków dostarcza mało, w większości bowiem *kradnie je* Javie. Wspominaliśmy o tym nie raz, zobaczmy w końcu, jak to wygląda. Rozważmy taką klasę:

```
public class MyWindow {
  private javax.swing.JFrame frame;
  private javax.swing.JLabel label;
}
```

Przedstawiony tu kod wygląda jak część aplikacji Swing języka Java i z pewnością skompilowałby się z użyciem kompilatora tego języka. Jest to jednocześnie pełnoprawny kod Grooviego! Nie musimy w jakiś tajemniczy sposób importować obiektów Javy lub obudowywać je jakimiś abstrakcjami (jak mia-

łoby to miejsce w przypadku niektórych języków skryptowych łączących się z Javą) – my po prostu używamy Javy. Nie zapominajmy, że na poziomie wykonania aplikacji (*runtime*) maszyna wirtualna nawet nie wie, że ma do czynienia z Groovim. Wszystko de facto jest Javą.

Głównym zadaniem Grooviego jest ułatwianie pracy programistom. W tym celu niektóre z klas standardowych języka Java zostały uzupełnione po stronie Grooviego o dodatkowe właściwości. Najwięcej przykładów możemy zobaczyć, operując na kolekcjach danych. Pracując np. ze standardowymi obiektami typu `ArrayList`, mamy do dyspozycji kilka metod, które ułatwiają przechodzenie po kolekcjach. Przykładem może tutaj być wspomniana wcześniej metoda `each`, wykonująca domknięcie dla każdego elementu listy. Co więcej, metoda ta operuje poprawnie nie tylko na kolekcjach, ale także np. dla napisów, pobierając w każdej iteracji kolejny znak. Oznacza to, że fizycznie wspomniane rozszerzenia możliwości obiektów standardowych zostały wykonane na poziomie klasy `Object`. Znajdziemy tu bardziej wyrafinowane operacje, niż przykładowa metoda `each`. Ich listę i opis przedstawiono w osobnej ramce.

Kolejnym rozszerzeniem standardowych klas języka Java jest dodanie dla niektórych z nich metod operatorowych. Przykładowo, mapy i listy mają zdefiniowany operator `[]`, dzięki któremu łatwo możemy odczytać lub zapisać wartość elementu pod wskazanym indeksem. Dodawanie nowych elementów do listy, obiektów typu `StringBuffer` czy konkatenacja stringów odbędzie się z użyciem operatora << (dla niektórych z klas zdefiniowano podobnie działający operator `<<=`, dokonujący jednocześnie przypisania).

Jedną z wielu praktycznych cech języka Java jest możliwość anonimowej implementacji interfejsów. Praktyka ta jest często stosowana przez programistów aplikacji Swing. Trzeba jednak przyznać, że przy implementacji dużej ilości np. obsługi zdarzeń, możemy łatwo wygenerować nieco rozdmuchany kod, którego lwią część będą stanowiły deklaracje, a faktyczny kod wykonujący jakieś działania będzie się gdzieś gubił. Groovy pozbawiony jest abstrakcji interfejsów, pozwala jednak na ich implementację z użyciem domknięć. Tak zapisany kod skupia się wyłącznie na swej funkcjonalności. Zobaczmy przykład implementujący zdarzenie dla obiektu `JButton`

```
JButton button = new JButton();
button.addActionListener( { event -> println
                        event.getSource() }
                        as ActionListener  )
```

Implementacja zdarzenia została zaimplementowana wprost, przy użyciu domknięcia. Jedynym dodatkowym elementem jest konieczność określenia, jak dane domknięcie powinno zostać zinterpretowane – zrobimy to z użyciem operatora `as`, którym wymuszamy konkretny typ klasy bądź interfejsu. Można zatem powiedzieć, że domknięcie zostanie skonwertowane do implementacji interfejsu `ActionListener`.

Tworzenie interfejsu użytkownika można jeszcze bardziej uprościć z użyciem budowniczych. Tę technikę opisano w kolejnym punkcie artykułu.

Programowanie z użyciem budowniczych

W poprzednich dwóch punktach dowiedzieliśmy się, jak korzystać z obiektów Javy w Groovim oraz jak wykorzystywać domknięcia. Ta wiedza, poparta odpowiednią praktyką, powinna umożliwić programistom Javy sprawną przesiadkę do nowego języka. Często jednak wiele problemów będziemy dalej rozwiązywać bez szczególnego zysku w kwestii ilości czasu bądź kodu. Bierze się to ze specyfiki niektórych zagadnień. Jednym z nich może być konstruowanie interfejsu użytkownika. Czy to w Groovim, czy w Javie, będziemy musieli stworzyć ciąg obiektów odpowiadających poszczególnym elementom okna, połączyć je w całość i zdefiniować zdarzenia. Zadanie nietrudne, lecz żmudne. Musimy bowiem zbudować strukturę o pewnej hierarchii, w której poszczególne elementy będą zawierały się w innych – podobnie jak na stronach HTML. Groovy, ze względu na swoją zgodność z Javą, pozwala przedstawione tu zagadnienia rozwiązać w sposób klasyczny, udostępnia też jednak znacznie szybszy mechanizm w postaci budowniczych dedykowanych konkretnym problemom. Nim zagłębimy się w szczegóły, przyjrzyjmy się przykładowi, nawet jeśli początkowo nie wszystko będzie tam zrozumiałe – Listing 1 prezentuje program konstruujący okno interfejsu użytkownika z informacją tekstową i przyciskiem.

To, co z pewnością zwraca na siebie uwagę w przykładzie, to oddanie kodem Grooviego hierarchii interfejsu użytkownika – bez pierwotnego tworzenia obiektów, a później sklejania ich w całość (miejscami może przypominać to nieco notację JSON). Mimo że kod wygląda dość niestandardowo, nie ma w nim żadnych nieznanych nam do teraz elementów składniowych. Wykorzystujemy tu dwie podstawowe cechy języka Groovy – możliwość inicjalizacji obiektów z użyciem map oraz anonimowe domknięcia. Skoro tak, możemy napisać własnych budowniczych do rozwiązywania konkretnych problemów – klasycznym przykładem może być tu narzędzie generujące zapytania SQL dla zadanych tabel. Zainteresowanych tematem odsyłam do lektury na temat klasy `groovy.util.BuilderSupport`, będącej bazą dla wszystkich budowniczych. Wcześniej jednak zachęcam do poeksperymentowania z budowniczymi dystrybuowanymi wraz z językiem. Mamy tu do wyboru następujące narzędzia:

- `NodeBuilder` – tworzy drzewiastą strukturę obiektów, po których można nawigować z użyciem GPath – prostego języka zapytań zbliżonego do Xpath;
- `DOMBuilder` – tworzy model dokumentu;
- `SAXBuilder` – obsługa dokumentu XML w ujęciu zdarzeniowym;
- `MarkupBuilder` – przydatny przy szybkiej budowie plików HTML i XML;
- `AntBuilder` – wsparcie dla dostępu i uruchamiania zadań narzędzia Ant;
- `SwingBuilder` – wsparcie dla budowy interfejsu użytkownika.

Obycie z budowniczymi wymaga początkowo odrobiny praktyki, z czasem staje się bardzo intuicyjne.

Zachęcam do przeanalizowania nieco większego przykładu z oficjalnej dokumentacji języka, w którym bezpośrednio z zapytania wysłanego do bazy danych budowany jest plik *XML*.

Metaprogramowanie w Groovim

Natura Grooviego pozwala na znacznie większą manipulację kodem w porównaniu do mechanizmu refleksji języka Java. Przede wszystkim mamy tutaj możliwość dodawania nowych elementów klas, a nie tylko ich dynamicznego uruchamiania. Współczesna Java dorobiła się co prawda narzędzi modyfikujących w czasie rzeczywistym kod bajtowy klas (służy do tego np. opisywana w SDJ biblioteka ASM, wykorzystywana nota bene wewnętrznie przez język Groovy), jednak w przypadku Grooviego mamy do czynienia z mechanizmem wbudowanym bezpośrednio w język.

Dynamiczne wywoływanie metod

Zacznijmy od małego przykładu ukazującego potęgę i prostotę dynamicznego wykonywania metod. Zaimplementujemy funkcję, która dla zadanego obiektu uruchomi metodę wskazaną za pomocą tekstu:

```
def runm(obj, mth, params) {
  obj."${mth}"(*params)
}
```

Przykład użycia tej funkcji mógłby wyglądać następująco:

```
runm("Hello world", "substring", [0,4])
```

Jak widzimy, Groovy pozwala nam posłużyć się dowolnym napisem w miejscu wywołania metody (lub odwołania do pola klasy). Prościej nie można już było tego zaimplementować!

Pewnego komentarza wymaga użyty w przykładzie sposób przekazywania parametrów do funkcji. Wywołanie `*params` zamienia listę na ciąg parametrów wejściowych funkcji. Bez tej konwersji przekazalibyśmy metodzie `substring` tablicę, która przecież nie jest akceptowanym typem wejściowym. Mechanizm ten można porównać do funkcji `apply` z języka JavaScript.

Sprawdzanie obecności pól i metod

W Javie każdy obiekt posiada atrybut `class`, dzięki któremu możemy zidentyfikować jego klasę. W języku Groovy mamy dodatkowo do dyspozycji właściwość `metaClass`, która umożliwia nam szybki dostęp do metamodelu obiektu.

Kod poniżej sprawdzi, czy obiekt `obj` klasy `MyClass` posiada pole o nazwie `age`:

```
MyClass.metaClass.hasProperty(obj, "age")
```

Właściwość `metaClass` zawiera w sobie listy dostępnych pól (`properties`) i metod (`methods`). Dzięki temu, poszukując konkretnych elementów klasy, możemy posługiwać się standardowymi operacjami na listach, np. z użyciem domknięć:

```
MyClass.metaClass.methods.findAll{ it.name.
                        startsWith("get") }
```

Pokazany przykład zwróci listę wszystkich metod, których nazwy rozpoczynają się od słowa *get*.

Modyfikacja prototypu klas

Groovy nie tylko pozwala nam przeglądać metamodel klas, ale również umożliwia jego modyfikację. Tym sposobem możemy rozbudować możliwości klas już istniejących, bez konieczności implementacji klas pochodnych. Zróbmy to dla standardowej klasy `String`:

```
String.metaClass.welcome = {
  "Welcome ${delegate}!"
}
```

Od tego momentu możemy posługiwać się metodą `welcome`:

```
print "Mel".welcome()
```

W pokazanym przykładzie zmienna `delegate` wskazuje na obiekt `String` sam w sobie. Nie możemy użyć tutaj wartości `this`, gdyż ta wskazywałaby na poziom wyżej – klasę, z której uruchamiamy kod (np. `ConsoleScript`). Jest to typowa cecha domknięć, a przecież dynamiczne metody implementujemy właśnie z ich użyciem.

W kwestii dynamicznego definiowania metod, Groovy pozwala nam na duże lenistwo. Jeśli metoda, którą chcemy zaimplementować, już istnieje, ale dla zupełnie innej (niespokrewnionej) klasy – możemy ją sobie zwyczajnie pożyczyć:

```
MyClass.metaClass.myMethod = obj.&method
```

Użyty tu operator `.&` jest wskaźnikiem na metodę, który w rezultacie zwraca domknięcie. Należy zwrócić uwagę, iż pożyczamy metodę konkretnego obiektu (wraz z jego kontekstem), a nie ogólną definicję metody z zadanej klasy.

Metamodel języka Groovy umożliwia nam również definiowanie nowych konstruktorów. Przykładowo, dla klasy `Person` zdefiniowanej następująco:

```
class Person {
  String name
}
```

utworzymy konstruktor przyjmujący jeden parametr, mający na celu inicjalizację pola obiektu:

```
Person.metaClass.constructor = {String n
   -> new Person(name: n) }
```

Zwróćmy uwagę, że wewnętrznie dokonujemy inicjalizacji obiektu z użyciem istniejącego konstruktora (w tym wypadku jest to domyślny konstruktor operujący na mapie). Nie jest to obowiązkowe, jednak często wygodne. Musimy jednak uważać, by nie odwołać się w takim kodzie ponownie do właśnie tworzonego konstruktora, uzyskamy bowiem w efekcie niechcianą rekurencję.

W każdym z przedstawionych wyżej przykładów deklarowaliśmy nową metodę lub konstruktor, bądź nadpisywaliśmy istniejące. Jeśli chcielibyśmy wykluczyć nadpisywanie, powinniśmy zamiast operatora równości posłużyć się operatorem przesunięcia bitowego w lewo (<<). W takiej sytuacji próba nadpisania istniejącej metody zakończyłaby się zgłoszeniem wyjątku.

Obiekty Expando

W niektórych przypadkach statyczna struktura klas może okazać się poważnym ograniczeniem. Typowym przykładem może być tutaj praca z bazami danych. Załóżmy, że chcemy pobrać z tabeli osób jedynie imiona i nazwiska. Czy będziemy przechowywać je w obiektach typu `Person`, z niezainicjalizowanymi pozostałymi polami, czy też obciążymy aplikację, wczytując niepotrzebne pola? Możemy ewentualnie użyć mapy, ale tracimy w tym układzie obiektową swobodę. Groovy wychodzi naprzeciw takim sytuacjom. Opisany wcześniej mechanizm dynamicznego rozbudowywania prototypów klas nie będzie tu zbyt użyteczny – w końcu nie chcemy modyfikować istniejących klas, a stworzyć nowe, zazwyczaj jedynie tymczasowo i w możliwie najprostszy sposób. Do takich celów służą obiekty `Expando`.

Programiści znający JavaScript wiedzą, że przypisanie obiektowi wartości (lub funkcji) dla nieistniejącego pola nie wygeneruje błędu, tylko to pole utworzy. Możemy zatem zacząć od stworzenia *pustego* obiektu (poprzez konstrukcję `new Object()` lub `{}`) ,a później, w zależności od potrzeby, dodawać nowe pola lub metody. Dokładnie na tej samej zasadzie działają obiekty `Expando` języka Groovy. Przykład poniżej rozwiązuje nasz problem z bazami danych - tworzy obiekt zawierający tylko interesujące nas pola:

```
def john = new Expando()
john.firstName = "John"
john.lastName = "Smith"
```

Zwróćmy uwagę, że nie odwołujemy się tutaj do metamodelu klasy (`metaClass`), tylko tworzymy pola *w locie*. Odwołanie do dowolnego elementu klasy jest zatem zawsze poprawne, nawet jeśli element ten nie istnieje. W najgorszym wypadku otrzymamy w rezultacie wartość `null`, jak na przykładzie poniżej:

```
print john.middleName
```

Nic nie stoi na przeszkodzie, by w ten sam sposób utworzyć metodę (oczywiście z użyciem domknięcia):

```
john.getFullName = {"$firstName $lastName"}
```

Należy jednak pamiętać, że obiekty `Expando` nie są typowymi obiektami w rozumieniu Javy. Po stronie tego języka są one reprezentowane nie jako klasyczne *java beans*, a jedynie jako elementy mapy obudowanej obiektem `groovy.util.Expando`.

Eval

Funkcja `eval` to poniekąd symbol większości języków dynamicznych od czasów powstania Lispa. Efektem jej działania jest wy-

konanie kodu, przekazanego funkcji w postaci napisu. Odpowiednik funkcji `eval` znajdziemy również w Groovim, w postaci statycznej klasy `Eval`. Jej implementacja w kompilowanym języku dla maszyny wirtualnej Java jest prawdziwym osiągnięciem (choć nie pierwszym i nie jedynym – patrz np. BeanShell lub Clojure). Zobaczmy przykładowy kod:

```
Eval.me('println "Hello dynamic world"')
```

Metoda `me` klasy `Eval` pobiera tekst w postaci parametru i wykonuje go jako kod programu. Chcąc sparametryzować wejściowy napis, tak, by w momencie wykonania mógł operować na aktualnych wartościach jakichś zmiennych, należy użyć dedykowanych metod klasy `Eval`, takich jak `x`, `xy`, `xyz` lub przeładowanej wersji metody `me`. Trzy pierwsze metody wprowadzają do ewaluowanego napisu zmienne (odpowiednio `x`, `y` i `z`). Ich użycie wygląda następująco:

```
Eval.xy(1, 2 ,'++x; x+y')
```

W pierwszej kolejności przekazujemy parametry, a następnie napis z kodem. Przeładowana metoda `me` działa podobnie, przyjmuje jednak jako parametr obiekt, który w napisie będzie rozpoznawany pod wskazaną przez nas nazwą.

Łącząc tę technikę z obiektami Expando, możemy przekazać wszystkie wartości, na których nam w danym momencie zależy, np.:

```
a = new Expando()
a.num = 5
Eval.me('obj', a, 'obj.num++')
```

Czas na bardziej wyrafinowany przykład demonstrujący użycie klasy `Eval` (oraz kilka innych cech języka). Listing 2 zawiera kod programu będącego rozbudowanym kalkulatorem, umożliwiającym rejestrację własnych funkcji. Metoda `calculate` klasy `DynCalc` po prostu wylicza przekazane wyrażenie matematyczne. Najbardziej interesujące rzeczy dzieją się jednak w metodzie `addFunction`, która pozwala zarejestrować dowolne wyrażenie w postaci składowej klasy `DynCalc`. Metoda przyjmuje trzy parametry – nazwę funkcji (`name`), wyrażenie reprezentujące funkcję (`fun`) oraz liczbę parametrów wejściowych funkcji (parametr `nr`, który może przyjmować wartości od 0 do 3). Metoda nie jest zbyt czytelna (ani skrajnie optymalna – założyłem sobie bardziej możliwie najkrótszy zapis z maksymalnym wykorzystaniem techniki ewaluacji), dobrze ukazuje jednak potęgę metaprogramowania w Groovim. Zastanówmy się, co dokładnie nasza metoda powinna robić? Chcąc zarejestrować nową funkcję, musimy ją wygenerować w postaci domknięcia, które będzie liczyło zadane wyrażenie. Napotykamy tu na pewne utrudnienie, polegające na zmiennej ilości parametrów wejściowych. W efekcie musimy, w zależności od ich liczby, skorzystać z różnych metod klasy `Eval` oraz wygenerować różną ilość wejść dla wynikowego domknięcia. W tym celu stworzyliśmy dwie listy (`evs` i `pas`), których elementy posłużą nam do skonstruowania opisanych elementów.

W uproszczeniu moglibyśmy rozwiązać nasz problem za pomocą domknięcia przyjmującego dwa parametry – wyrażenie i tablicę wartości wejściowych, które następnie zostaną (dzięki operatorowi `*`) skonwertowane do ciągu niezależnych parametrów. Takie domknięcie (stworzone w celach pomocniczych) reprezentowane jest przez zmienną `ee` na listingu (zwróćmy uwagę na sposób wywołania metody klasy `Eval`). Ostatnim krokiem będzie zatem utworzenie wynikowego domknięcia, które przyjmie dokładnie zadaną liczbę parametrów wejściowych, a wewnętrznie użyje opisanego wyżej domknięcia pomocniczego. Nie mamy innej możliwości, jak zbudować to domknięcie w postaci łańcucha znaków (patrz zmienna `clo` w przykładzie). Dzięki temu przyjmie ono na wejściu zadaną liczbę parametrów. Ponadto we wnętrzu operuje na zmiennych `x` i `y`, których wartości przekażemy z użyciem metody `Eval.xy`. `x` w tym wypadku wskazuje na utworzone wcześniej domknięcie pomocnicze (`ee`) a `y` – na wyrażenie. Rezultatem wykonania `Eval.xy` będzie dynamiczne utworzenie oczekiwanego przez nas domknięcia, które finalnie zostanie przypisane dla klasy `DynCalc` w postaci nowej metody.

Nie jest to łatwy do czytania przykład, jednak pokazane dalej sposoby jego użycia powinny jasno zobrazować finalny efekt i tym samym możliwości języka. Zwróćmy uwagę, że tworzona funkcja `parabolic`, obliczająca położenie obiektu po zadanym czasie w rzucie ukośnym, zawiera nie tylko operacje matematyczne, ale również deklaracje zmiennych wewnętrznych. Stworzyliśmy zatem w kilkunastu liniach kodu kalkulator o bardzo rozbudowanych możliwościach.

Romans z Groovim dla uwikłanych w Javę

Pisaliśmy wcześniej o używaniu klas Javy z poziomu języka Groovy. Łatwo możemy sobie wyobrazić konieczność sytuacji odwrotnej. Groovy z definicji ma ułatwiać pisanie kodu. Wielu programistów, zwłaszcza pracujących od dawna nad jakimiś aplikacjami, nie przepisze ich teraz na nowy język.

Mogą jednak pokusić się o napisanie fragmentów kodu w Groovim i późniejsze uruchomienie takich podprogramów z poziomu Javy. Przyda się to zwłaszcza w sytuacjach, w których Groovy pozwoli na znaczne oszczędzenie czasu. Operacje na bazach danych, plikach *XML* czy elementach interfejsu użytkownika mogą tu być dobrym przykładem.

Zacznijmy jednak od czegoś mniejszego – prostej klasy języka Groovy:

```
class Test {
  private String name
  public def value
}
```

Jest to klasa specyficzna dla Grooviego, zawiera bowiem niezdefiniowany typ, mogący przyjmować dowolne wartości. Wspominaliśmy już jednak, że maszyna wirtualna Javy nie jest w stanie rozpoznać, że ma do czynienia z innym językiem. Kompilując powyższy kod, otrzymamy zatem klasyczny plik *.class*. A *import* naszej klasy do Javy będzie wyglądał następująco:

```
Test test = new Test();
```

Mam nadzieję, że nikt już nie ma wątpliwości, że Groovy to Java, a integracja języków jest zupełnie przezroczysta. Właśnie to czyni ten język silnym w zestawieniu z innymi dynamicznymi językami dedykowanymi maszynie wirtualnej Javy. Możemy w Groovim napisać całą bibliotekę i dostarczyć ją programistom Javy – dla nich nie będzie żadnej różnicy, poza koniecznością dołączenia dedykowanych językowi plików *jar*.

Zobaczmy teraz, jak zachowują się po stronie Javy rozwiązania specyficzne dla języka Groovy. Dla pełnego zrozumienia obu światów musimy bowiem wiedzieć, jak poszczególne mechanizmy są mapowane pomiędzy językami. W przykładzie powyżej nie określiliśmy dla zmiennej `value` typu i nie zainicjalizowaliśmy jej żadną wartością, która mogłaby ten typ determinować. Rozwiązanie nie jest po stronie Javy niczym tajemniczym – `value` jest określane poprzez `java.lang.Object`. Gdyby w klasie `Test` miała miejsce inicjalizacja (np. napisem lub liczbą), przypisalibyśmy obiektowi wartość konkretnego typu, który przecież zawsze będzie dziedziczył po klasie `Object`. Groovy zatem jedynie wy-

korzystał coś, co od zawsze było elementem Javy. Różnica polega na braku mocnej kontroli typów i konieczności rzutowania.

Przyjrzyjmy się teraz ogólnej konstrukcji klas. Często pisząc małe przykładowe programy, zazwyczaj z użyciem konsoli Groovy, nie deklarujemy żadnych klas, co jest przecież niedopuszczalne w Javie. W takiej sytuacji tworzymy niejawnie klasę domyślną, która po skompilowaniu przyjmie nazwę pliku, w którym napisaliśmy nasz kod – zostanie on umieszczony wewnątrz metody `main`. Każda funkcja napisana bezpośrednio w pliku stanie się oczywiście metodą (niestatyczną) wygenerowanej klasy.

Groovy pozwala na zadeklarowanie dowolnej ilości klas (również publicznych) w obrębie jednego pliku. Wszystkie zostaną skompilowane do standardowych klas Java. Nie ma natomiast możliwości tworzenia klas wewnętrznych – zarówno w obrębie klasy, jak i jej metod. Składnia języka nie pozwala na takie konstrukcje i powinny być one zastępowane domknięciami. Każde domknięcie jest natomiast traktowane przez Javę jako klasa wewnętrzna, rozszerzająca `groovy.lang.Closure`. Tym oto sposobem te dziwne konstrukcje są osiągalne z poziomu języka Java. By je uruchomić, skorzystamy z opisanej już wcześniej metody `call`, występującej w trzech wersjach - bez parametrów, z jednym parametrem typu `Object` lub listą parametrów tego typu. Wynikiem działania metody jest zawsze obiekt typu `Object`.

Kolejnym specyficznym elementem języka Groovy są obiekty `Expando`. Po stronie Javy reprezentowane one są poprzez instancje klasy `groovy.util.Expando`, która de facto jest prostym obudowaniem mapy. Każdy dynamicznie tworzony element obiektu `Expando` zostanie zatem zapisany w wewnętrznej mapie, pod podaną przez nas nazwą. Metody obiektów `Expando` reprezentowane są przez domknięcia, zatem utworzone zostaną dla nich klasy wewnętrzne.

Odwołanie się do elementów obiektów `Expando` nie może po stronie Javy zatem odbywać się bezpośrednio poprzez nazwy, a jedynie z użyciem dedykowanych metod: `setProperty` i `getProperty`. Na przykładzie poniżej pokazano, w jaki sposób w Javie wykonać metodę takiego obiektu:

```
((groovy.lang.Closure)((groovy.util.Expando)
    myExpando).getProperty("method")).ca
                                ll();
```

Dalsza lektura

Nie było technicznej możliwości, by w tak krótkim artykule przedstawić wszystkie możliwości rozbudowanego języka programowania. To temat na książkę. W artykule zdołaliśmy zaledwie wyłożyć podstawy języka – jego składnię i obszary możliwych zastosowań, bez szczególnego zagłębiania się w wyjaśnienia, jak poszczególne mechanizmy zostały zaimplementowane w Javie.

Pominęliśmy w całości standardową bibliotekę Grooviego (GDK), której klasy w pełni wykorzystują możliwości języka, pozwalając programować szybciej, niż z użyciem standardowych klas Javy (zwłaszcza dzięki budowniczym). Znajdziemy tu przykładowo wsparcie dla obsługi baz danych, pracy z dokumentami XML czy obsługi aplikacji okienkowych. Programistów aplikacji internetowych zachęcam do zainteresowania się, jak pisać servlety (a właściwie – groovlety) w języku Groovy oraz jak wydajnie można połączyć aplikacje tego języka ze Springiem.

Jeszcze ciekawszy może okazać się webowy szkielet Grails, w całości dedykowany szybkiemu wytwarzaniu aplikacji internetowych w języku Groovy. Temu tematowi poświęciliśmy osobny artykuł w niniejszym wydaniu SDJ, do którego lektury serdecznie zachęcam.

Pobierając pakiet instalacyjny Grooviego, otrzymujemy wraz z binariami kompletny podręcznik języka. Poruszanie się po nim początkowo jest dość niewygodne, a pierwsze rozdziały nie skupiają się na łagodnym wprowadzeniu do języka. Z czasem można jednak się przyzwyczaić do struktury książki i wiele się z niej nauczyć. Zwolenników klasycznych podręczników zachęcam do przeczytania świetnie napisanej książki – *Groovy in Action* (red. *Dierk König*). Jej lektura jest prawdziwą przyjemnością.

Autorzy zakładają, że czytelnik wie coś o programowaniu (zwłaszcza w Javie), dzięki czemu książka nie jest przepełniona banałami.

Podsumowanie

Groovy jako język dedykowany maszynie wirtualnej Java nie jest jedynym rozwiązaniem. Spośród alternatyw warto wspomnieć na przykład o Clojure, języku stworzonym na bazie Lispa, kompilującym się bezpośrednio do kodu binarnego Javy. Nie należy też zapominać o przeniesieniu do Javy języków dobrze znanych i sprawdzonych – jak choćby Ruby czy Python. W tych ostatnich przypadkach integracja z istniejącymi klasami i bibliotekami Java nie jest jednak już tak prosta i wbudowana w język jak w przypadku Grooviego. Groovy powstał jako szybsza (w zapisie) i bardziej dynamiczna notacja dla Javy, pozostając z nią nierozerwalnie związanym. Możliwość naprzemiennego pisania kodu w Javie lub Groovim i jego łatwa integracja z całą pewnością należą do najmocniejszych stron języka.

Język Groovy początkowo potrafi zachwycić ilością konstrukcji i ułatwień, jednak przy pierwszej próbie napisania większego programu możemy odnieść wrażenie pewnego braku spójności poszczególnych koncepcji. Język czerpie w zauważalny sposób ze sprawdzonych rozwiązań takich jak Python, Perl, Ruby lub JavaScript, jednak połączenie tych światów, wraz z koniecznością zachowywania całkowitej zgodności z Javą, powoduje, że programowanie w Groovim nie zawsze jest intuicyjne, przynajmniej w początkowej fazie pracy z językiem. Na szczęście to tylko pierwsze wrażenie. Szybko można się oswoić z co dziwniejszymi konstrukcjami języka i zacząć bardziej sprawne programowanie. Duża dowolność Grooviego potrafi dać programiście sporo satysfakcji z tworzonego kodu, a przede wszystkim umożliwia znacznie bardziej zwięzłe pisanie w porównaniu z Javą. W przypadku małych lub średnich projektów, warto rozważyć ich implementację w całości w Groovim. Dla dużych aplikacji, przy zauważalnej rotacji programistów w zespole, podlegająca lepszej kontroli Java może być bezpieczniejszym rozwiązaniem. Tym niemniej warto delegować do Grooviego pojedyncze zadania, z którymi ten język poradzi sobie w znacznie łatwiejszy sposób.

W Sieci

- *http://groovy.codehaus.org* – strona domowa języka;
- *http://grails.org* – strona webowego szkieletu dla języka Groovy;
- *http://jcp.org/en/jsr/detail?id=241* – strona poświęcona standaryzacji języka Groovy (JSR 241);
- *http://www.stelligent.com/content/view/44/71* – interesujący wywiad z Guillaumem Laforge, liderem prac nad rozwojem języka.

DAVID DE ROSIER

Programista-podróżnik. Pasjonat WEB 2.0, estetycznego kodowania i algorytmów szachowych. Zawodowo – szkoleniowiec i specjalista od aplikacji JEE i MDA w sektorze finansowym.
Kontakt z autorem: ddrosier@gmail.com

WS-BPEL

Instrumentacja procesów biznesowych

Web Services Business Process Execution Language (WS-BPEL) jest językiem programowania, którego celem jest opisanie procesów zachodzących między partnerami biznesowymi. W artykule zostanie przedstawiona przykładowa procedura BPEL, korzystająca z jednego z ogólnodostępnych w Internecie Web serwisów.

Dowiesz się:
- Jak działają synchroniczne procesy BPEL;
- Jak używać następujących czynności:
  - Invoke;
  - Assign;
  - ForEach;
  - If;
  - Scope;
- Czym jest PartnerLink;
- Jak zadeklarować i użyć tzw. 'predicate';
- Jak testować i debugować proces BPEL.

Powinieneś wiedzieć:
- Znać w stopniu podstawowym NetBeans;
- Czym jest:
  - EAI,
  - ESB,
  - SOA,
  - wsdl,
  - xsd.

Poziom trudności

Za pomocą języka BPEL możemy pisać programy, które są Web serwisami, i które korzystają z innych już istniejących Web serwisów, czyli pozwalają na integrowanie różnych usług. Język ten pozwala między innymi na utworzenie nowej usługi, która agreguje usługi już istniejące, lub pozwala na konwersję danych dostarczanych przez partnerów biznesowych lub tworzy nowy interfejs do usług już istniejących. Zadeployowany proces BPEL jest widziany jako nowy Web Service. Kod źródłowy programu napisanego w języku BPEL ma postać XML'a, pozwala to na używanie dowolnego edytora tekstowego do pisania procedur BPEL, jednak praktycznie jest stosować edytory przystosowane do tego celu.

Dla zilustrowania procedury BPEL wybrałem usługę sieciową, która dostarcza dwie operacje: pierwsza, dla zadanego kraju zwraca listę największych miast oraz druga, dla zadanego kraju i miasta zwraca aktualna pogodę. Celem tego artykułu jest zarysowanie możliwości procedur BPEL w środowisku NetBeans dlatego też przyjmuję, że czytelnik zna to środowisko oraz jest zorientowany w technologiach związanych z architekturą zorientowaną na usługi (SOA).

W literaturze anglojęzycznej często spotyka się termin *orchestration* – odpowiednikiem w języku polskim jest *instrumentacja*. Za słownikiem muzycznym PWM: instrumentacja jest to układ utworu muzycznego na zespół instrumentalny lub sporządzenie partytury. Porównanie do muzyki jest celne, gdyż program BPEL opisuje interakcje między procesami biznesowymi, operacje na danych otrzymywanych od partnerów biznesowych oraz łączy wiele usług w jedną całość.

Szczegóły dotyczące specyfikacji BPEL można znaleźć na portalu OASIS.

Niezbędne środowisko

Aby kontynuować i móc w praktyce przetestować działanie przykładowej procedury BPEL, musimy spełnić następujące wymagania:

- niezbędne jest połączenie z Internetem;
- należy zainstalować najnowszą wersję NetBeans z portalu *http://www.netbeans.org/downloads/*. Ważne jest, aby była to wersja ALL (Rysunek 1), gdyż tylko ta wersja wspiera BPEL w wersji 2.0.
- Krok opcjonalny: pisząc ten artykuł, korzystałem z wersji 6.5.1 NetBeans, dlatego też musiałem wykonać upgrade biblioteki http-binding, ponieważ ta, która była w w.w. instalacji zawierała błąd.

- *BPEL* – skrót od Web Services Business Process Execution Language (WS-BPEL), jest to standard języka, który opisuje w krokach interakcje między usługami sieciowymi.
- *WSDL* – skrót od Web Services Description Language, jest to język który opisuje Web Serwisy – opisuje protokoły, operacje, formaty danych w usługach sieciowych.
- *SOA* – skrót od service-oriented architecture. Jest to podejście architektoniczne, którego główna ideą jest integracja procedur biznesowych, powtarzalnych zadań biznesowych.
- *EAI* – skrót od Enterprise Application Integration polega na integracji zestawu aplikacji komputerowych przy użyciu rozwiązań software'owych.
- *ESB* – skrót od Enterprise Service Bus, jest to 'apparatus', którego implementacje, między innymi, dostarczają narzędzi pozwalających na transformację danych, rutowanie komunikatów, zmniejszenie ilości połączeń point-to-point, itd.
- *XSD* – skrót od XML Schema Document. Jest to dokument zawierający metadane, opisuje w sposób abstrakcyjny zasady, które musi spełniać dokument XML aby móc zostać uznany z poprawny.
- *XPATH* – skrót od XML Path Language. Jest to język zapytań służący do wybierania pojedynczych węzłów lub ich grup z danego dokumentu XML wg zadanego kryterium.

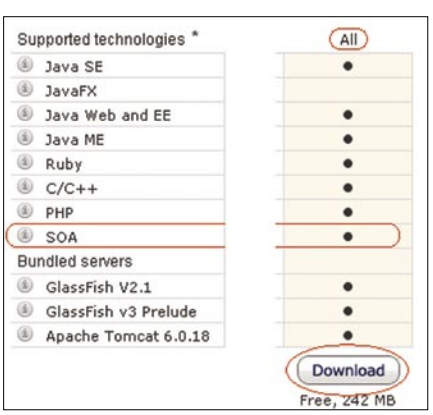
Procedura instalacji tej biblioteki jest następująca:

- na wstępie należy ściągnąć plik *httpbc.jar* (url znajduje się w części *W sieci*);
- następnie należy uruchomić serwer Glassfish v2 i zalogować się na jego konsolę jako administrator;
- w końcu w lewym panelu wybrać *JBI->Components*, zaznaczyć *sun-http-binding* i kliknąć *Upgrade*, i w następnym okienku wskazać ściągnięty plik. Kroki zobrazowano na Rysunkach 2, 3 i 4.

Scenariusz

W celu ilustracji podstawowych możliwości BPEL wykonamy prosty proces. W Internecie można znaleźć wiele publicznie dostępnych usług webowych. Jedną z nich jest Global Weather.
Usługa ta dostarcza dwie operacje:

- `GetCitiesByCountry` – dostarcza listę miast – wymaga jednego parametru: nazwa kraju.
- `GetWeather` – dostarcza aktualną pogodę – wymaga dwu parametrów: nazwę kraju i nazwę miasta.



Rysunek 1. *Pobranie odpowiedniej wersji NetBeans.*

Rysunek 2. *Upgrade biblioteki http-binding część 1*

Proces, który wykonamy dostarczy nam informację o aktualnej pogodzie w głównych miastach zadanego kraju, np. dla Polski.

Operacja, którą zaimplementujemy, będzie oczekiwała jednego parametru – nazwy kraju, i na tej podstawie wywoła operację `GetCitiesByCountry`, a następnie dla każdego miasta, w pętli wywoła operację `GetWeather`. Wynikiem działania naszego procesu będzie XML zawierający dane o aktualnej pogodzie dla największych miast np. w Polsce. Przykładowy fragment danych zwracanych przez ww operacje znajduje się na Listingu 1. Jak widać zadaniem procedury BPEL będzie, po wywołaniu operacji `GetCitiesByCountry`, wybranie wszystkich tagów `CITY` i dla każdej znalezionej wartości wywołanie operacji GetWeather łącząc odpowiedzi ostatniej operacji w jedną całość i zwrócenie jej jako parametr wyjścia naszej procedury BPEL.

Szybka ścieżka

Realizacja przykładu została podzielona na dwanaście kroków.

- Krok 1 – Początek projektu.
  - Tworzymy nowy projekt w NetBeans
  - naciskamy [*Ctrl+Shift+N*];
  - wybieramy kategorię Samples/SOA, typ projektu Synchronous BPEL process (Rysunek 5);

Listing 1. *Wyniki działania operacji GetCitiesByCountry oraz GetWeather*

```
<string>
<NewDataSet>
  <Table>
    <Country>Poland</Country>
    <City>Warszawa-Okecie</City>
  </Table>
…..itp.....
zaś fragment odpowiedzi operacji GetWeather to:
<CurrentWeather>
  <Location>Warszawa-Okecie, Poland (EPWA) 52-10N 020-58E 107M</Location>
  <Time>Jun 15, 2009 - 03:30 AM EDT / 2009.06.15 0730 UTC</Time>
  <Wind> from the W (270 degrees) at 10 MPH (9 KT) (direction variable):0</Wind>
  <Visibility> greater than 7 mile(s):0</Visibility>
  <Temperature> 68 F (20 C)</Temperature>
  <DewPoint> 53 F (12 C)</DewPoint>
  <RelativeHumidity> 59%</RelativeHumidity>
  <Pressure> 29.97 in. Hg (1015 hPa)</Pressure>
  <Status>Success</Status>
</CurrentWeather>
```

Listing 2. *Deklaracje zmiennych*

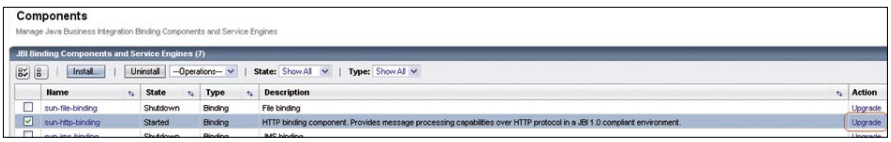
```
<variable name="GetCitiesByCountryOut"
                xmlns:tns="http://www.webserviceX.NET"
                messageType="tns:GetCitiesByCountrySoapOut"/>
<variable name="GetCitiesByCountryIn"
                xmlns:tns="http://www.webserviceX.NET"
                messageType="tns:GetCitiesByCountrySoapIn"/>
```

Listing 3. *Konwersja odpowiedzi partnera na listę miast*

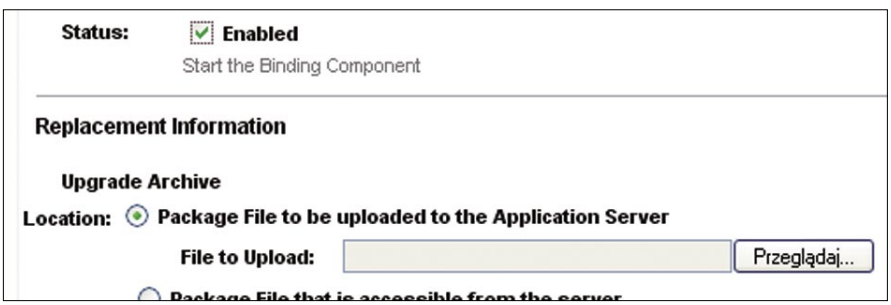
```
<assign name="Assign2">
   <copy>
      <from>
            sxxf:doUnMarshal($GetCitiesByCountryOut.parameters/ns3:GetCitiesByCoun
                  tryResult)
      </from>
      <to>$miasta.resultType/ns2:paramA</to>
   </copy>
   <copy>
      <from>$miasta.resultType/ns2:paramA//City</from>
      <to>$miasta.resultType/ns2:paramA</to>
   </copy>
</assign>
```

- wciskamy przycisk *Next* i wpisujemy nazwę projektu, np. `aktualnaPogoda`, i wciskamy przycisk *Finish*.

Wynikiem tego kroku jest utworzenie dwu modułów, jeden zawierający proces BPEL, drugi to tzw. *Composite Application*. *Composite Application* jest projektem, który gromadzi jeden lub więcej innych modułów i który zostanie deployowany na serwerze Java Business Integration (JBI), w naszym przypadku będzie to GlassFish v2.



Rysunek 3. *Upgrade biblioteki http-binding część 2*



Rysunek 4. *Upgrade biblioteki http-binding część 3*

Listing 4. *Warunek pomijający brak danych*

```
<if name="If1">
   <condition>not(contains($GetWeatherOut.parameters/ns3:GetWeatherResult,
                           'Data Not Found'))
   </condition>
   <assign name="Assign4"/>
</if>
```

Listing 5. *Konwersja odpowiedzi partnera biznesowego*

```
<assign name="Assign4">
   <copy>
      <from>
           sxxf:doUnMarshal($GetWeatherOut.parameters/ns3:GetWeatherResult)
      </from>
      <to>$outputVar.resultType/ns2:paramA[$ForEach1Counter]</to>
   </copy>
```

Listing 6. *Fragment powodujący błąd budowania aplikacji, który należy zakomentować bądź usunąć*

```
<!--wsdl:port name="GlobalWeatherSoap12" binding="tns:GlobalWeatherSoap12">
      <soap12:address location="http://www.webservicex.net/globalweather.asmx" />
    </wsdl:port>
    <wsdl:port name="GlobalWeatherHttpGet" binding="tns:GlobalWeatherHttpGet">
      <http:address location="http://www.webservicex.net/globalweather.asmx" />
    </wsdl:port>
    <wsdl:port name="GlobalWeatherHttpPost" binding="tns:GlobalWeatherHttpPost">
      <http:address location="http://www.webservicex.net/globalweather.asmx" />
    </wsdl:port-->
```

Listing 7. *Deklaracja parametru WE dla testów procedury BPEL*

```
<soapenv:Body>
   <syn:typeA>
      <syn:paramA>Poland</syn:paramA>
   </syn:typeA>
</soapenv:Body>
```

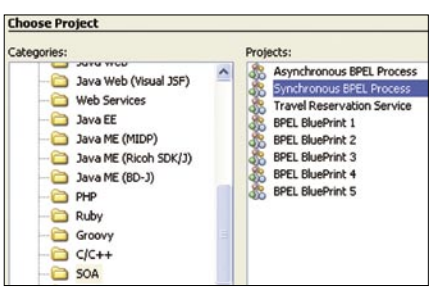
Ponieważ korzystamy z przykładowego projektu jako podstawę, którą rozbudujemy, środowisko automatycznie utworzyło plik WSDL dla naszej nowej usługi oraz plik xsd. Automatycznie utworzone pliki WSDL oraz XSD są bardzo ubogie lecz na potrzeby tego artykułu w zupełności wystarczające. Oczywiście w rzeczywistej aplikacji biznesowej pliki te należy uzupełnić do własnych potrzeb lub utworzyć nowe własne.

- Krok 2 – Utworzenie lokalnej kopii dokumentu opisującego zewnętrznego partnera biznesowego.

Należy na module BPEL z menu kontekstowego wybrać pozycję *new external WSDL document*. Wybieramy *from URL* i wpisujemy adres WSDL, który nas interesuje, w naszym przypadku będzie to *http://www.webservicex.net/globalweather.asmx?wsdl* jak na Rysunku 6. Efektem tego kroku jest pojawienie się nowego pliku WSDL, jak na Rysunku 7. W ten sposób, lokalnie posiadamy model opisujący usługę webową naszego partnera biznesowego.

- Krok 3 – przypisanie partnerów biznesowych w procedurze BPEL.

Otwieramy plik z rozszerzeniem *.bpel* w trybie *Design*. W ten sposób uruchomiony zostanie graficzny edytor procedur BPEL. Metodą *drag and drop* przeciągamy nowy plik WSDL na prawe pole dla Partner Link, jak zaznaczono na czerwono na Rysunku 8. Po przeciągnięciu pliku WSDL, otworzy się okno *Create new Partner Link* – pozostawiamy wartości domyślne. Czynności te można oczywiście wykonać ręcznie, dokonując odpowiednich wpisów w pliku źródłowym procedury BPEL. W celach edukacyjnych dobrym pomysłem jest obserwowanie zmian zachodzących w kodzie źródłowym edytowanej procedury podczas edycji w trybie graficznym.



Rysunek 5. *Wybór nowego projektu*

W katalogu *Process Files* pojawi się nowy plik WSDL o nazwie *globalweather.asmxWrapper*. Jest to wrapper, w którym został zdefiniowany `partnerLinkType`. PartnerLinkType jest kanałem komunikacji między stronami. Komunikacja ta może odbywać się w dwie strony. Należy zauważyć, że wrapper ten zawiera klauzule importującą WSDL, który został utworzony w kroku drugim. `PartnerLinkType` reprezentuje interakcje między naszym procesem a partnerem biznesowym.

Patrząc na proces BPEL w trybie *Design* (Rysunek 9), widzimy z lewej strony `PartnerLink`, który reprezentuje klienta wysyłającego dane wejściowe – w naszym przykładzie jest to nazwa kraju – i w odpowiedzi otrzymuje listę z aktualną pogodą. Z prawej strony zaś mamy partnera dostarczającego listę miast dla zadanego kraju oraz aktualną pogodę dla zadanego miasta. W części środkowej jest nasz proces, który będzie odpowiednio manipulował danymi.

- Krok 4 – edycja procedury BPEL.

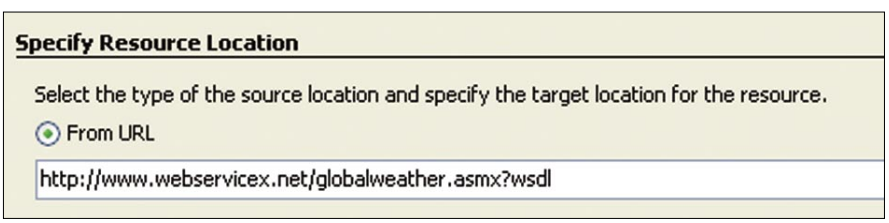
Z palety czynności przeciągamy czynność `Invoke` i wstawiamy ją poniżej czynności `Assign1`. Następnie przeciągamy czynności `Assign`, `ForEach`, `Scope`, `Assign`, `Invoke`, `If`, `Assign`. Na czynności `Invoke1` znajduje się koperta, którą należy teraz przeciągnąć do operacji `GetCitiesByCountry`. Kopertkę z czynności `Invoke2` przeciągamy do operacji `GetWeather`. Powinniśmy otrzymać proces taki jak na Rysunku 10. Krótki opis czynności znajduje się w Ramce.

Tym sposobem otrzymaliśmy zgrubny szkic procedury BPEL. Nie należy się przejmować komunikatami błędów gdyż te zostaną usunięte w następnych krokach.

- Krok 5 – tworzymy zmienne

W tym kroku utworzymy zmienne: dwie o zasięgu globalnym i dwie o zasięgu Scope1. W tym celu dwukrotnie klikamy na czynność `Invoke1`, naciskając przyciski *Create* utworzymy zmienne wejścia i wyjścia. Analogicznie postępujemy z czynnością `Invoke2` (patrz Rysunek 11). Zmienne te zawierać będą parametry wywołania oraz wartości zwrócone przez partnerów biznesowych.

Jeśli otworzymy kod źródłowy naszej procedury, w sekcji `<variables>` zobaczymy deklarację tych zmiennych, będą to zmienne `GetCitiesByCountryOut` oraz `GetCitiesByCountryIn`, których deklaracja mają postać jak na Listingu 2 oraz zmienne `GetWeatherOut` i `GetWeatherIn`. Należy zwrócić uwagę na miejsce deklaracji tych zmiennych. Dwie ostanie zmienne są widocznie wyłącznie w Scope1 w przeciwieństwie do dwu pierwszych, które mają zasięg globalny.



Rysunek 6. *Pobranie WSDL partnera biznesowego*

Listing 8. *Fragment odpowiedzi od partnera biznesowego*

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                                    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/ http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <ns1:typeA xmlns:msgns="http://localhost/SynchronousSample/SynchronousSample"
                        xmlns:ns1="http://xml.netbeans.org/schema/SynchronousSample">
      <ns1:paramA>
        <Location xmlns="">Gdansk-Rebiechowo, Poland (EPGD) 54-23N 018-28E 138M</Location>
        <Time xmlns="">May 25, 2009 - 11:30 AM EDT / 2009.05.25 1530 UTC</Time>
        <Wind xmlns=""> from the E (090 degrees) at 10 MPH (9 KT):0</Wind>
        <Visibility xmlns=""> greater than 7 mile(s):0</Visibility>
        <Temperature xmlns=""> 60 F (16 C)</Temperature>
        <DewPoint xmlns=""> 46 F (8 C)</DewPoint>
        <RelativeHumidity xmlns=""> 58%</RelativeHumidity>
        <Pressure xmlns=""> 30.09 in. Hg (1019 hPa)</Pressure>
        <Status xmlns="">Success</Status>
      </ns1:paramA>
```

Opis czynności

W literaturze anglojęzycznej używa się słowo *Activity*, wydaje się że odpowiednikiem w języku polskim najlepiej oddającym sens tego słowa jest *czynność*.

- Czynność `Invoke` jest używana do wywoływania usług webowych oferowanych przez partnerów.
- Czynność `Assign` posiada jedną lub kilka operacji kopiowania między zmiennymi zadeklarowanymi w procedurze BPEL.
- Czynność `ForEach` iteruje sekwencyjnie N razy po szeregu zadeklarowanych czynności.
- Czynność `Scope`. Każda czynność `ForEach` musi zawierać czynność `Scope`, jest to wymagane przez specyfikację OASIS. Scope pozwala na tworzenie struktury zagnieżdżonych hierarchii.
- Czynność `If` jest to struktura znana z większości języków programowania i pozwala na wybranie jednej gałęzi czynności na podstawie dostarczonego warunku.

- Krok 6 – zasilanie parametru wejścia partnera biznesowego

Dwukrotnie klikamy na `Assign1`, otwierając edytor mapowania (*Mapper*). Usuwamy strzałkę łączącą lewą kolumnę z prawą, następnie przeciągamy `paramA` (z lewej strony edytora `Mapper Variables/inputVar/inputType/paramA`) na `CountryName` (z prawej strony edytora `Mapper Variables/GetCitiesByCountryIn/parameters/CountryName`), tak jak na Rysunku 12.

Tym sposobem kopiujemy zawartość parametru wejściowy naszej procedury do parametru wejścia procedury GetCitiesByCountry partnera biznesowego. Innymi słowy wstawiamy nazwę kraju jako parametr operacji GetCitiesByCountry.

- Krok 7 – utworzenie zmiennej tymczasowej.

Teraz utworzymy zmienną, która będzie przechowywała listę miast, którą otrzymamy od naszego partnera biznesowego. W tym celu wklejamy

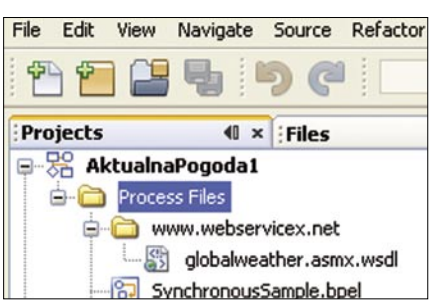
```
<variable name="miasta" messageType="ns1:
                        responseMessage"/>
```

do pierwszego bloku `<variables>` – są to zmienne globalne. To samo możemy *wyklikać* w okienku Navigator procesu, w widoku logicznym BPEL, w menu kontekstowym pozycji *Variables* wybrać *add Variable*.

Następnie odszukujemy w kodzie źródłowym czynność `Assign2` i podmieniamy tę czynność na tę przedstawioną na Listingu 3. Czynność ta spowoduje wyekstrahowanie zawartości tagu <City> i skopiowanie jej do zmiennej *miasta*, otrzymując wyłącznie listę miast bez zbędnych danych.

Ponadto do deklaracji procesu (tag `<process>`) wklejamy:

```
xmlns:sxxf="http://www.sun.com/wsbpel/2.0/
   process/executable/SUNExtension/
                        XPathFunctions"
```



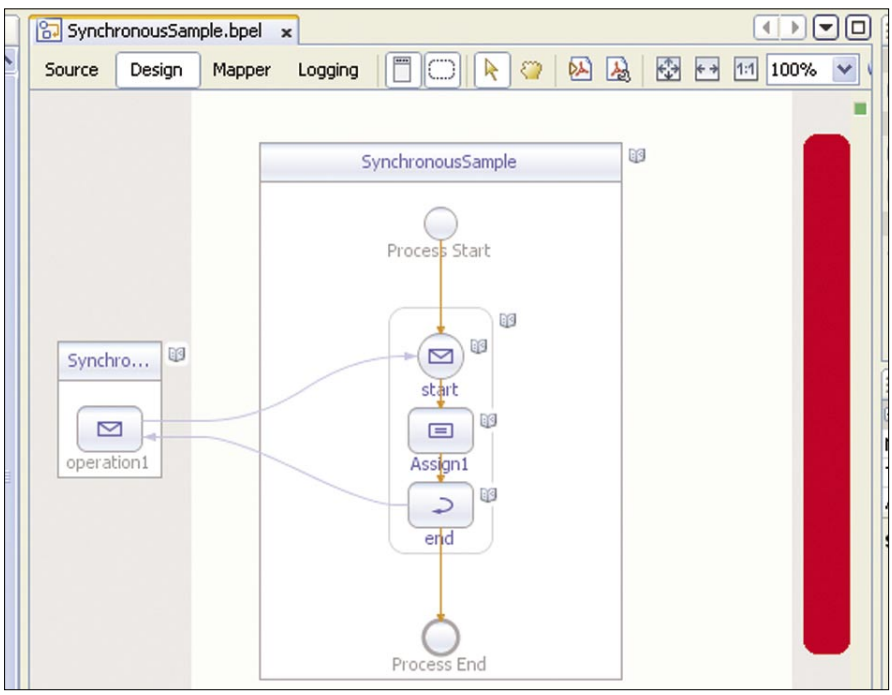
Rysunek 7. *WSDL partnera biznesowego*

To spowoduje, że predefiniowane funkcje XPath będą widoczne dla naszej procedury BPEL-owej.

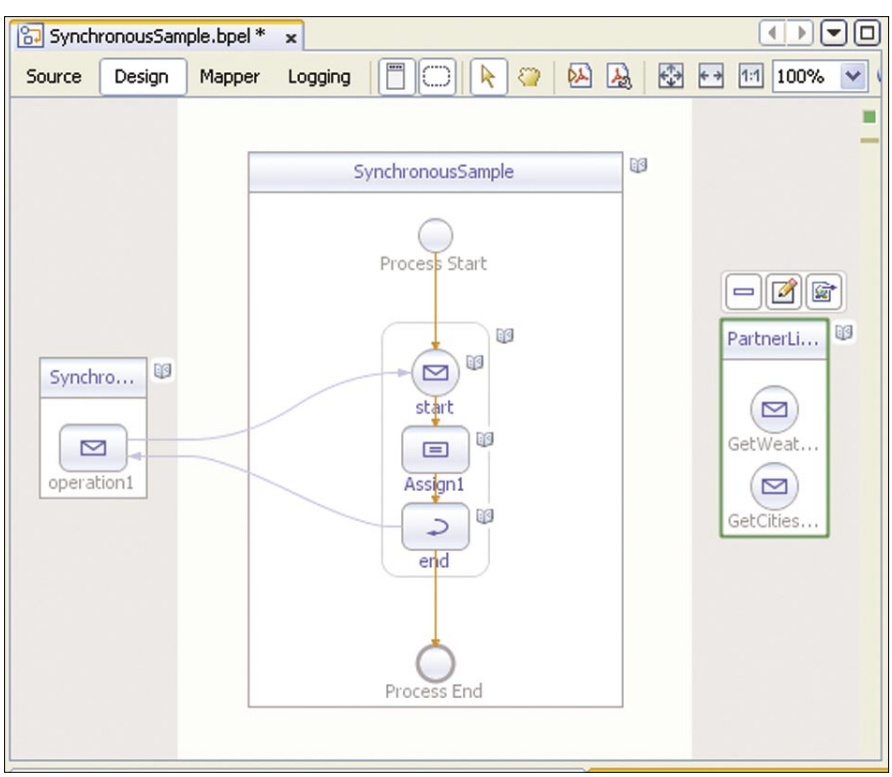
Bez tego wpisu funkcja `doUnMarshal` byłaby niewidoczna i procedura BPEL nie kompilowałaby się. Dostępne funkcje Xpath są widoczne w zakładce Mapper edytora graficznego BPEL, jeśli metodą *drag and drop* przeciągniemy funkcję *BPEL/doUnMarshal*, to tag `<process>` samoczynnie uzupełni się niezbędnym wpisem.

W czynności `Assign2` odpowiedź partnera biznesowego zostanie skonwertowana do XML-a, następnie przepisana do zmiennej miasta, na koniec przy pomocy XPath zostaną wybrane wyłącznie tagi o nazwie `City`.

Teraz wciskamy [*Alt+Shift+F9*], co uruchomi walidację pliku – do czynności



Rysunek 8. *Pusty proces BPEL*



Rysunek 9. *Dodany zewnętrzny PartnerLink*

`Assign2` włącznie nie powinno być żadnych błędów.

- Krok 8 – inicjalizacja pętli

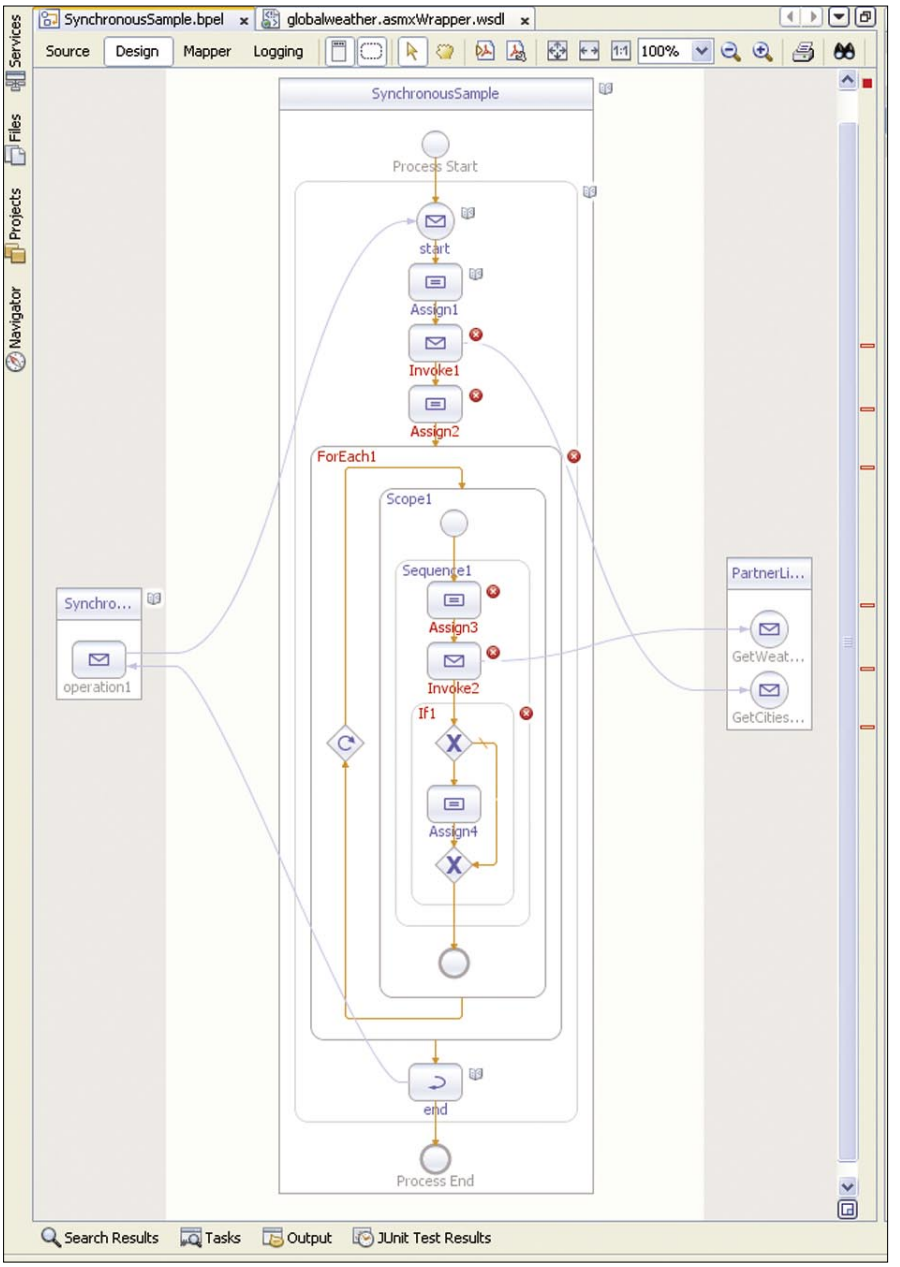
W tym kroku ustalimy ilość iteracji czynności ForEach. W widoku *Design* naszego procesu BPEL w menu kontekstowym czynności `ForEach1` wybieramy *Properties*. W pole *Start Counter Value* wpisujemy 1 – wartość początkowa licznika. Mając zaznaczony `ForEach1`, wciskamy przycisk *Mapper*. Zaznaczamy pozycję *Final Value*, przeciągamy funkcję *Count* z grupy *Node*, łączymy *paramA* z *Final Value;* funkcja *Count* zwraca ilość węzłów, w naszym przypadku jest to ilość miast w liście otrzymanej z operacji `GetCitiesByCountry`, ponadto dołączamy stałą *1* do *Start Value*, uzyskując obraz taki jak na Rysunku 13. Dla uproszczenia przyjmujemy, że wywołana usługa zwróci przynajmniej jedno miasto; pominiemy walidację poprawności wpisanej nazwy kraju. Spójrzmy w kod źródłowy deklaracji ForEach: `<finalCounterValue>count( $miasta.resultType/ns2:pogoda[.])</ finalCounterValue>` w celu lepszego zrozumienia działania mappera edytora graficznego zwrócić uwagę w jaki sposób jest wywoływana funkcja count.

- Krok 9 – ustanowienie *Predicate*

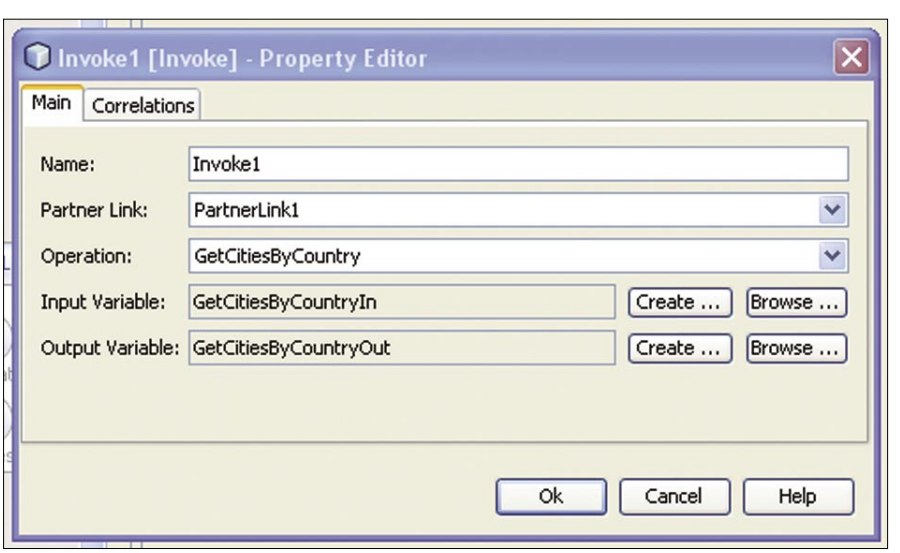
Predicate jest to warunek, który jest nakładany na węzeł xml'owy zawierający wiele pozycji. Wynikiem jest podzbiór zawierający tylko te węzły, które spełniają ten warunek. W naszym przykładzie *predicate* użyjemy do pobierania kolejno jednej pozycji z listy miast. Predicate można tez wykorzystać do dodawania nowych węzłów w liście. W kroku tym utworzymy tzw. *Predicate*, który pozwoli nam na iterację po poszczególnych miastach, jako licznik wykorzystamy zmienną `ForEach1Counter`. Po pierwsze, dwukrotnie klikamy na `Assign3`, następnie kilkamy na `paramA` w menu kontekstowym gdzie jedną z opcji jest *Add Predicate*. Po wybraniu tej opcji pojawi się nowe okienko, w którym przeciągamy zmienną na prawo do `Predicate`, tak jak widać na Rysunku 14, i natychmiast przypisujemy zmienne wejścia do partnera biznesowego tak jak na Rysunku 15.

- Krok 10 – walidacja odpowiedzi partnera biznesowego

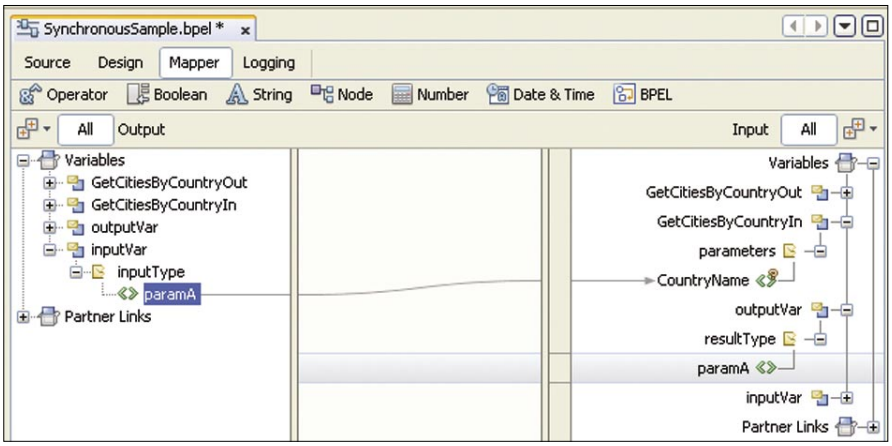
Nasz partner biznesowy w sytuacji, gdy nie znajdzie danych o pogodzie dla wskazanego miasta, zwraca łańcuch `Data Not Found`. Z



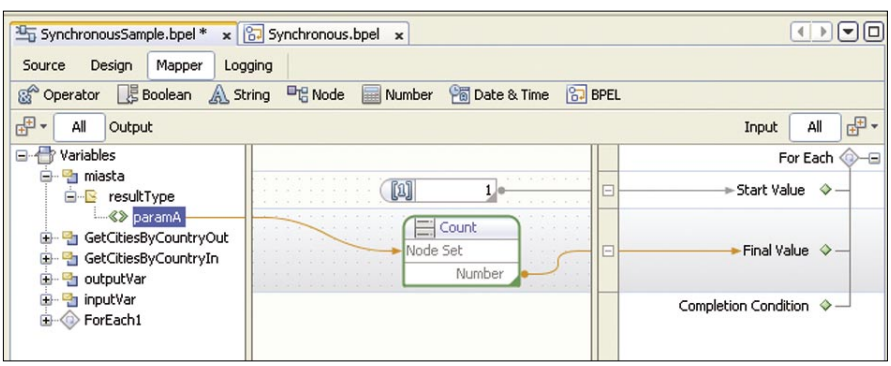
Rysunek 10. *Szkic procesu BPEL*



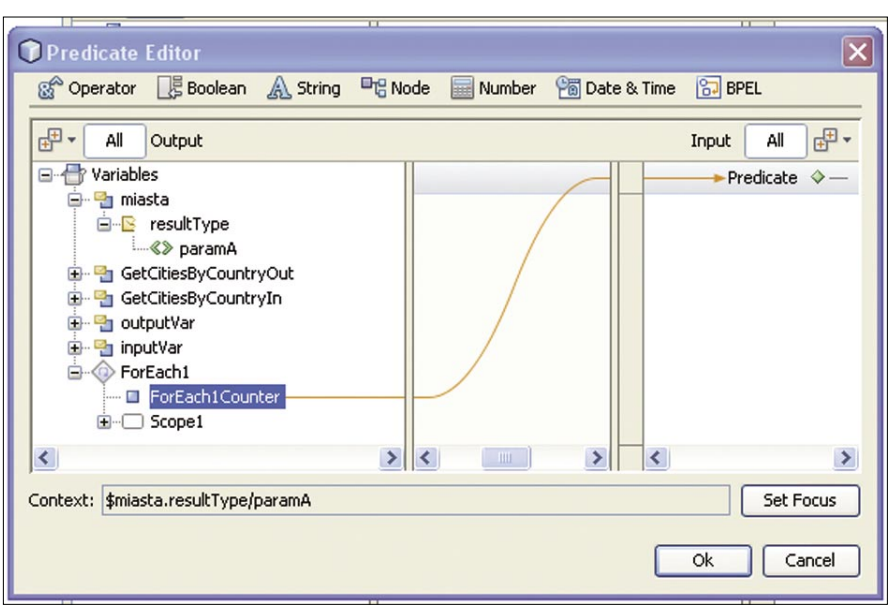
Rysunek 11. *Zmienne Wejścia/Wyjścia dla partnera biznesowego*



Rysunek 12. *Mapowanie*



Rysunek 13. *Wartości początkowa i końcowa ForEach*



Rysunek 14. *Iteracja po Miastach*

tego względu, niezbędne jest dodanie klauzuli warunkowej, która pozwoli na pominięcie tych miast. W naszym przykładzie jest to `If1`. Należy uzupełnić czynność `IF` o warunek `<condition>`, uzyskując kod przedstawiony na Listingu 4.

Podobny efekt jak na Listingu 4 można uzyskać, korzystając z edytora graficznego. Warunek ten sprawdza zawartość odpowiedzi operacji `GetWeather`. Jeśli w odpowiedzi nie znajduje się łańcuch znaków `'Data Not Found'`, zostanie wykonana czynność Assign4, w innym wypadku program rozpocznie kolejną iterację.

- Krok 11 – tworzenie listy miast z aktualną pogodą

W czynności `Assign4` będziemy kopiowali odpowiedź partnera do parametru wyjścia naszej procedury. W każdej iteracji, do parametru *Wyjścia* zostanie dodana pogoda dla danego miasta. Aby uzyskać ten efekt należy czynność `Assign4` zmodyfikować o treść z Listingu 5.

Również stworzyliśmy `predicate (paramA[$ForEach1Counter])` dla `paramA`, lecz tym razem ręcznie. Odpowiedź partnera biznesowego skonwertujemy do XML-a, a następnie skopiujemy do `outputVar`, który jest parametrem wyjścia naszego procesu. W ten sposób w każdej iteracji będzie dodawana jedna pozycja do parametru wyjścia.

W tym miejscu spotyka nas przykrość, gdyż wersja NetBeans, z której korzystałem, zawierała błąd w mechanizmie budowania plików *.casa*. Aby obejść ten błąd, należy usunąć bądź zakomentować cytowany na Listingu 6 fragment w pliku *globalweather.asmx.wsdl*.

Wprowadzając zmianę przedstawioną na Listingu 6, usunęliśmy porty, z których nie korzystamy i które powodują błędne budowanie aplikacji. Prawdopodobnie w kolejnych wersjach NetBeans ten błąd zostanie usunięty.

Teraz otwieramy plik *.casa* (Service Assembly modułu `AktulnaPogoda1Application`), czyścimy i budujemy (*Clean and build*) oba moduły: `AktualnaPogoda1` oraz `AktulnaPogoda1Application`. W edytorze graficznym

W Sieci

- Specyfikacja WS-BPEL OASIS – *http://www.oasis-open.org/*;
- Tutorial WS-BPEL *http://www.eclipse.org/tptp/platform/documents/design/choreography_html/tutorials/wsbpel_tut.htm*;
- NetBeans – *http://www.netbeans.org/downloads/*;
- EAI – *http://en.wikipedia.org/wiki/Enterprise_Integration_Patterns*;
- Projekt open-ESB *https://open-esb.dev.java.net/*;
- biblioteki http-binding – *http://download.java.net/jbi/binaries/open-jbi-components/main/nightly/latest/ojc/httpbc.jar*;
- Globalweather web-service *http://www.webservicex.net/globalweather.asmx?wsdl*;
- Wyszukiwarka publicznie dostępnych usług webowych *http://seekda.com/.*

plików casa otrzymujemy obraz identyczny do tego z Rysunku 16. Na Rysunku 16 zobrazowano, jak działa nasza przykładowa procedura, jak jest wywoływana i jakie usługi wywołuje.

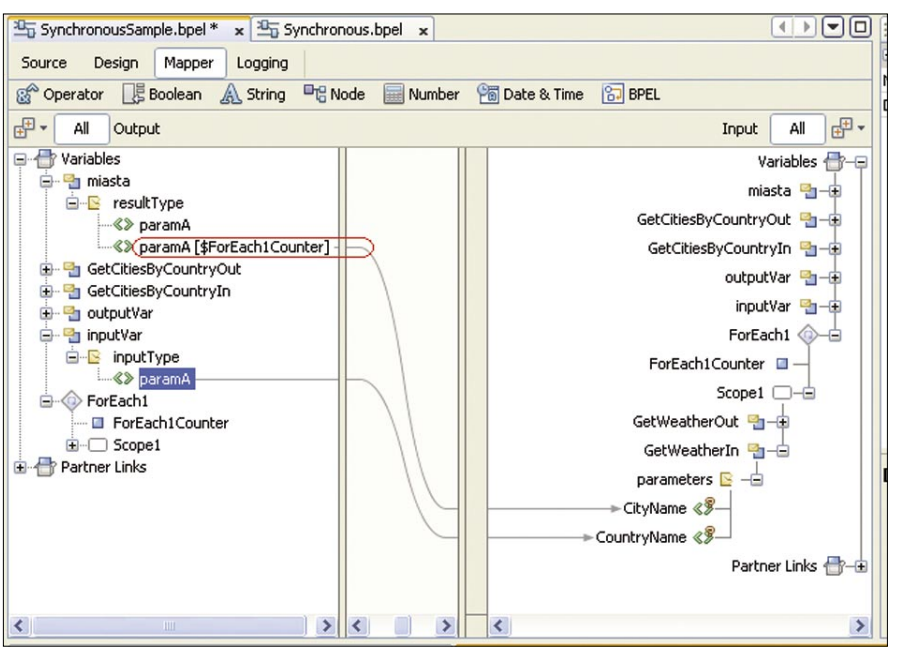
- Krok 12 – uruchomienie procedury BPEL.

W tym kroku przetestujemy naszą procedurę. Przechodzimy do zakładki *Services*, wybieramy *Servers/GlassFish v2* i w menu kontekstowym wybieramy *Start*. Gdy serwer się uruchomi, otwieramy plik *Input*, znajdujący się w zakładce *Projects* w module `AktualnaPogoda1Application` w folderze *Test/TestCase0*, tak jak zilustrowano na Rysunku 17. W pliku tym wpisujemy *Poland* w miejsce parametru `paramA`, tak jak widać na Listingu 7.

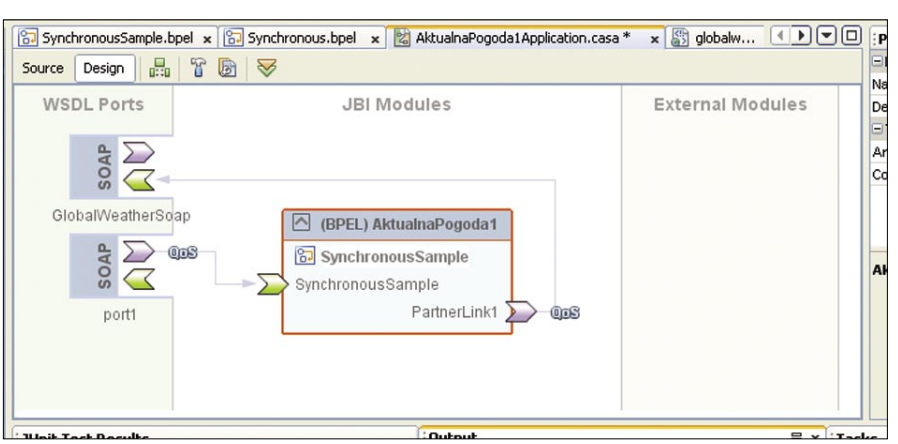
Następnie w menu kontekstowym w module `AktualnaPogoda1Application` wybieramy *Clean and Build*, dalej *Deploy*, a na koniec w menu kontekstowym *TestCase0* wybieramy *Run*. Gdy NetBeans odpyta nas, gdzie zadeployować aplikację, należy wskazać serwer Glassfish v2. Po chwili powinien pojawić się nowy plik w folderze *TestCase0* o nazwie *aktualna data z dopiskiem Failed* i czerwoną kopertą. Otwieramy ten plik. Powinien on zawierać treść bardzo podobną do tej z Listingu 8. Listing 8 zawiera fragment odpowiedzi otrzymanej od partnera – w rzeczywistości plik ten jest długi, oczywiście dane otrzymane będą się różnić od tych z listingu, gdyż pogoda zmienną jest. Pomimo że procedura BPEL-owa zadziałała bezbłędnie, przy wyniku testu pojawiła się adnotacja Failed; w tym przypadku jest to bez znaczenia, gdyż mechanizm testujący oczekuje, aby odpowiedź była identyczna z zawartością pliku *Output*, co jest niemożliwe, gdyż pogoda bezustannie się zmienia.

W dowolnym kroku procedury BPEL w edytorze graficznym można dodać breakpoint poprzez wskazanie wybranej czynności i naciśnięcie [*CTRL + F8*]. Pozwala to na analizowanie przebiegu procedury oraz obserwację zmiennych. Oczywiście przypadek testowy należy uruchomić poprzez wybranie *Debug* w menu kontekstowym *TestCase0*. Możliwość debugowania procedur BPEL w środowisku NetBeans jest bardzo rozbudowana i ergonomiczna co znakomicie pomaga w pisaniu procedur.

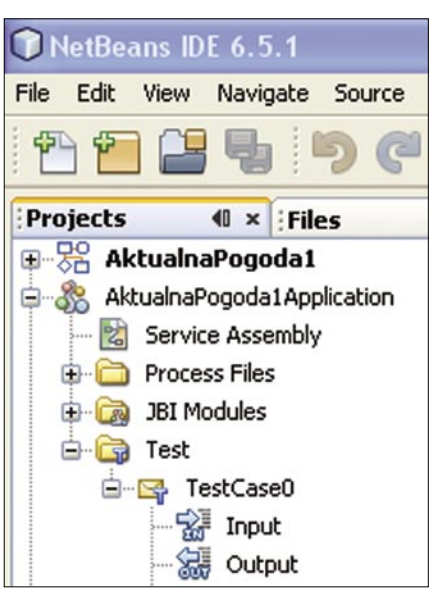
Funkcjonalność SOA, dostarczona w środowisku NetBeans, jest bardzo ergonomiczna i łatwa do opanowania. Narzędzie to można z powodzeniem wykorzystać też w sytuacji gdy np. tworzymy aplikacje korporacyjną w postaci skomplikowanego interfejsu użytkownika, który zbiera duże ilości danych od operatora a następnie ma za zadanie wysłać je do odległego Web service'u do którego nie mamy dostępu w czasie produkcji i posiadamy jedynie jego WSDL. W takiej sytuacji, na potrzeby produkcji musimy utworzyć atrapę takiej usługi sieciowej; atrapę która będzie musiała umieć wykonać walidacje biznesowe, zwracać różne kody błędów, tak abyśmy mogli testować naszą aplikację. W takiej sytuacji utworzenie zestawu procedur BPEL wydaje się optymalne, dużo bardziej praktyczne niż pracowicie pisać Web serwisy np. bezpośrednio w javie co jest czasochłonne i może prowadzić do sytuacji gdzie programiści czekają jeden na drugiego. Używając BPEL w kilka minut można tworzyć i modyfikować odpowiednie procedury BPEL tak aby z łatwością wykonywać scenariusze testujące wszystkie przypadki użycia.



Rysunek 15. *Przypisanie aktualnego miasta do parametru WE partnera biznesowego*



Rysunek 16. *Moduły JBI*



Rysunek 17. *Testowanie – parametr wejściowy*

PIOTR ZERYNGER

Autor jest konsultantem JEE i zajmuje się językiem Java od momentu jego powstania, czyli od 1995 roku.

Kontakt z autorem: z_piotr@yahoo.com

Język skryptowy Lua

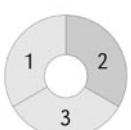
Charakterystyka i rola języka skryptowego Lua w programowaniu gier (i nie tylko)

Lua jest jednym z najczęściej wykorzystywanych języków skryptowych w programowaniu gier komputerowych. Skrypty często służą implementacji algorytmów logiki gry, SI czy interface'u użytkownika, co ułatwia modyfikację i rozbudowę aplikacji. Przyjrzymy się charakterystyce języka Lua oraz integracji skryptów Lua w aplikacjach takich jak gry komputerowe.

Dowiesz się:
- Co to jest Lua;
- Jak praktycznie korzystać z Lua.

Powinieneś znać:
- Podstawy programowania w C/C++;
- Filozofię języków skryptowych.

Poziom trudności



Języki skryptowe to języki programowania umożliwiające rozszerzanie funkcjonalności aplikacji komputerowych bez konieczności zmiany kodu wynikowego. Są one również intensywnie wykorzystywane przy automatyzacji działań administracyjnych systemu, np. w powłoce systemu Unix. Głównym założeniem tego typu języków jest możliwość ingerencji użytkownika w istniejący i działający system bez znajomości kodu źródłowego. Przykładami takich języków są Bash, Python, Perl, PHP, QuakeC, GameMonkey, UnrealScript oraz Lua.

Języki skryptowe często wykorzystywane są w procesie tworzenia gier komputerowych w celu sterowania przebiegiem fabuły (np. misje, dialogi), podejmowania decyzji w algorytmach sztucznej inteligencji (np. sterowanie postaciami NPC, ang. *non-playable characters*) czy też graficznej obsługi interfejsu gry (GUI, ang. *graphical user interface*). Jednym z najczęściej wybieranych przez twórców gier języków skryptowych jest język Lua, który został użyty w tytułach takich jak: Crysis, FarCry, Baldur's Gate, S.T.A.L.K.E.R., Wiedźmin czy World of Warcraft.

Lua jest lekkim językiem skryptowym łączącym ze sobą elementy prostego języka proceduralnego (składnia podobna do języka Pascal) z bardziej złożonymi konstrukcjami opisu danych opartych na tablicach asosjacyjnych, a także rozszerzalną semantyką. Lua jest projektem open-source'owym, zaimplementowanym w C zgodnie ze standardami ANSI C, którego głównym celem jest łatwość użycia, prostota, wydajność i przenośność kodu.

Niniejszy artykuł ma na celu przybliżenie czytelnikowi, poprzez liczne przykłady, charakterystyki języka Lua, oraz wskazanie praktycznych aspektów wykorzystania języków skryptowych w aplikacjach komputerowych.

Na początku omówimy podstawowe elementy składni języka Lua, a następnie przejdziemy do integracji skryptów z aplikacjami napisanymi w C/C++. Zapoznamy się również z wbudowanymi standardowymi bibliotekami Lua ułatwiającymi pisanie bardziej skomplikowanych skryptów.

Opis języka – podstawy

Jak przekonamy się w dalszej części tekstu, składnia Lua jest zwięzła i intuicyjna dla każdego, kto miał jakąkolwiek styczność z językami programowania.

Najczęściej przygodę z nowym językiem programowania rozpoczyna się od napisania najprostszego programu typu 'hello world'. Taki program w Lua przedstawiono w Listingu 1.

Konwencje leksykalne

Identyfikatory w Lua mogą składać się z dowolnego łańcucha znaków alfanumerycznych lub znaku _, jednak nie mogą rozpoczynać się cyfrą. Istnieją również słowa zastrzeżone, tj. `and`, `or`, `break`, `else` itp., które nie mogą opisywać identyfikatorów. Lua rozróżnia wielkość liter, zatem `for` oznacza pętlę, natomiast `For` widziany jest jako identyfikator. Podwójny myślnik `--` rozpoczyna komentarz i ignoruje na etapie kompilacji/wykonywania wszystkie znaki występujące po nim aż do końca linii. Istnieje również możliwość zakomentowania całego bloku składającego się z kilku linii poprzez `--[[" i "]]`.

Typy i wyrażenia

W Lua nie deklarujemy typów zmiennych – każda zmienna ma dynamiczny typ określany przez jej wartość.

Istnieje 8 podstawowych typów danych w Lua:

- `nil` – przyjmuje jedną wartość `nil`, która określa brak konkretnej wartości (często `nil` jest stosowany do określenia zmienną niezainicjalizowaną);
- `boolean` – przyjmuje standardowe wartości boolowskie, czyli `true` lub `false`;
- `number` – liczba reprezentująca wartości rzeczywiste (zmiennoprzecinkowe); liczby możemy zapisywać na różne sposoby, np.: 18, 3.141592, 1.23e12, 4.18e-3;
- `string` – standardowy łańcuch znaków, np. `"What is your name?"`;
- `userdata` – typ danych umożliwiający przetrzymywanie dowolnych danych z języka C (np. wskaźników) w strukturach Lua;
- `function` – funkcje zdefniniowane jako wykonywalny ciąg instrukcji; funkcje mogą być deklarowane wewnątrz innych funcji, mogą być argumentami

innych funkcji, lub być zwracane z innej funkcji;
- `thread` – wątek w Lua – z uwagi na charakter niniejszego artykułu nie będziemy omawiać wątków w Lua;
- `table` – tablica asosjacyjna będąca najcięższym orężem Lua; dokładny opis tablic znajduje się w dalszej części.

W dowolnym miejscu skryptu możemy sprawdzić typ danej zmiennej poleceniem `type` (jak pokazano w Listingu 2).

Lua, jak przystało na porządny język programowania, oferuje również różnego rodzaju operatory do definiowania wyrażeń. Wyróżniamy:

- operatory arytmetyczne, czyli `+` (dodawanie), `-` (odejmowanie), `*` (mnożenie), `/` (dzielenie), `%` (modulo) i `^` (potęgowanie);
- operatory porównania, czyli `==` (równe), `~=` (różne), `<` (mniejsze), `>` (większe), `<=` (mniejsze lub równe), `>=` (większe lub równe);
- operatory logiczne, takie jak `and` (część wspólna), `or` (suma logiczna), `not` (negacja).

Instrukcje i funkcje

Lua obsługuje standardowy zestaw instrukcji podobnych do tych stosowanych w języku Pascal czy C. Istnieje możliwość definiowania bloków instrukcji objętych słowami kluczowymi `do` i `end`, które można kontrolować poprzez `break` czy `return`. Na listę pojedynczych instrukcji składają się:

- instrukcje przypisania `=`;
- instrukcje warunkowe, `if/elseif/else/end`;
- pętle, `for/do/end`, `while/do/end`, `repeat/until`;
- wywołania funkcji;
- lokalne deklaracje zmiennych.

Instrukcja przypisania może dotyczyć kilku zmiennych. Pojedyncze instrukcje mogą opcjonalnie kończyć się średnikiem.

Listing 3 ilustruje wykorzystanie funkcji, zastosowanie pętli oraz instrukcji warunkowych na przykładzie funkcji wyznaczającej wartość minimalną i maksymalną z trzech liczb, oraz funkcji obliczającej pierwiastki równania kwadratowego.

Jak widzimy w przykładzie, funkcje mogą przyjmować i zwracać więcej niż jeden parametr. Istnieje również możliwość deklarowania funkcji rekurencyjnych (Listing 4).

Tablice

Tablice są najistotniejszymi strukturami danych w Lua. Są one reprezentowane przez kolekcję par (klucz, wartość), gdzie kluczem może być wszystko poza wartością nil. Elementami tablicy mogą być wszystkie wartości podstawowych typów

Listing 1. *Program typu 'hello world' w Lua wraz z przykładem komentarzy*

```
-- first.lua
print("Hello from Lua!");
--[[
   Przykład komentarza
   umieszczonego w kilku liniach
]]
```

Listing 2. *Sprawdzanie typu zmiennych w Lua*

```
print(type(a))   --> nil   (zmienna 'a' nie została zainicjalizowana)
a = 10
print(type(a))   --> number
a = "string test"
print(type(a))   --> string
a = print          -- a staje się funkcją print
a(type(a))         --> function
```

Listing 3. *Przykład pętli, funkcji i instrukcji warunkowych*

```
-- wyznaczanie wartości minimalnej i maksymalnej
min_max = function(a, b, c)
   local tab = {a, b, c};
   local idx_min = 1;
   local idx_max = 1;
   for i=2,3 do
      if tab[i]<tab[idx_min] then idx_min = i; end;
      if tab[i]>tab[idx_max] then idx_max = i; end;
   end;
   return tab[idx_min], tab[idx_max];
end;
print(min_max(1,8,3));      --> 1   8
-- rozwiązywanie równania kwadratowego
quadratic_eq = function(a, b, c)
   delta = b*b-4*a*c;
   if delta>0 then
      sqr_delta = math.sqrt(delta);
      x0 = (-b-sqr_delta) / (2*a);
      x1 = (-b+sqr_delta) / (2*a);
      return x0, x1;
   elseif delta==0 then
      x = -b/(2*a);
      return x, x;
   else
      return nil, nil;
   end;
end;
print(quadratic_eq(1, -1, -2));   --> -1   2
```

Listing 4. *Przykład funkcji rekurencyjnej w Lua*

```
-- obliczenie n!
function factorial (n)
   if n == 1 then
      return 1;
   else
      return n * factorial(n-1);
   end
end
-- obliczenie 5!
print(factorial(5));   --> 120
```

Lua, czyli także same tablice. Dzięki temu mechanizm tablicy umożliwia definiowanie użytkownikowi dowolnych struktur danych, które mogą być utożsamiane z elementami takimi jak: *lista*, *wektor*, *kolejka*, *zestaw* czy *stos*.

Listing 5. *Przykłady wykorzystania tablicy w Lua*

```
-- prosty wektor liczb
squares = {1, 4, 9, 16, 25, 36, 49, 64, 81}
-- zbiór danych
creature = { name = "Ghoul", health = 100 }
print(creature["name"]);    --> Ghoul
print(creature.name);         --> Ghoul
-- lista
list = nil;
list = {next = list, value = 2};
list = {next = list, value = "test"};
                    list = {next = list, value = math.pi};
ptr = list;
while ptr do
   print(ptr.value);
   ptr = ptr.next;
end;          --> 3.141592   test   2
-- struktura/klasa
Vec3 = {};
Vec3.new = function(x, y, z)
   return {x=x, y=y, z=z, length=function() return math.sqrt(x*x+y*y+z*z) end;}
end;
v = Vec3.new(4, 2, 4);
print(v:length());       --> 6
```

Listing 6. *Inicjalizacja maszyny wirtualnej i wykonanie skryptu Lua*

```
#include <iostream>
/// dołączenie nagłówków Lua
extern "C"
{
#include "lua.h"
#include "lualib.h"
#include "lauxlib.h"
}
/// dołączenie biblioteki Lua
#pragma comment (lib, "lua51.lib")

/// program główny
int main()
{
   /// stworzenie maszyny wirtualnej Lua
   lua_State* s = lua_open();
   /// inicjalizacja standardowych bibliotek
   luaL_openlibs(s);
   /// wykonanie skryptu Lua
   if (luaL_dofile(s, "sdj.lua"))
   {
      /// błąd
      std::cout << "Error while running script: " << lua_tostring(s, -1) << std::
                     endl;
      return 1;
   }
   /// zamknięcie maszyny wirtualnej
    lua_close(s);

   return 0;
}
```

W tradycyjnych językach programowania jakimi są C czy Pascal w większości przypadków złożone struktury danych reprezentowane są właśnie przez dynamiczne listy lub wektory prostych typów danych. Tablice w Lua umożliwiają implementację o wiele bardziej złożonych i elastycznych struktur, stworzonych specjalnie na potrzeby rozwiązania danego zagadnienia. Listing 5 pokazuje przykłady wykorzystania tablic w Lua realizując proste zbiory danych, listę jednokierunkową, czy strukturę zawierającą zarówno atrybuty jak i metody.

Integracja z C++

Po zapoznaniu się z podstawowymi elementami języka skryptowego Lua przejdźmy do jego praktycznego zastosowania w aplikacjach napisanych w C++. Listing 6 obrazuje kompletny kod programu w C++, którego celem jest poprawne zainicjalizowanie maszyny wirtualnej i wykonanie skryptu Lua.

W pierwszym rzędzie należy dołączyć nagłówki Lua (*lua.h*, *lualib.h* i *lauxlib.h*) oraz skompilowaną bibliotekę Lua (w przykładzie jest to *lua51.lib*). Binarną wersję biblioteki musimy sobie sami zbudować ze źródeł Lua, lub skorzystać z gotowej skompilowanej wersji umieszczonej na stronie domowej projektu.

Następnie poleceniem `lua_open()` tworzymy maszynę wirtualną (`lua_State`) i rejestrujemy standardowe biblioteki opisane w dalszej części artykułu. Teraz możemy już wykonać skrypt poleceniem `luaL_dofile()`, sprawdzając jednocześnie, czy skrypt nie zawiera błędu (np. składniowego). Na zakończenie program zwalnia maszynę wirtualną poleceniem `lua_close()`.

Wiemy już, jak wykonać skrypt na maszynie wirtualnej Lua z poziomu kodu aplikacji w C++. W powyższym przykładzie brakuje jednak symbiozy pomiędzy aplikacją a skryptem – każdy z tych procesów wykonuje się *dla samego siebie*. Musimy zatem stworzyć mechanizm komunikacji pomiędzy aplikacją a skryptem.

Mechanizmem umożliwiającym sterowanie naszą aplikacją jest zarejestrowanie funkcji aplikacji w maszynie wirtualnej Lua, która jest punktem łączącym obydwa procesy. Załóżmy, że chcemy udostępnić maszynie wirtualnej funkcję aplikacji, która przyjmuje dwie liczby i łańcuch znaków jako parametry wejściowe i dodatkowo zwraca wynik. Funkcję w naszej aplikacji zamieściliśmy w Listingu 7.

W pierwszej fazie pobieramy argumenty funkcji, odwołując się do elementów na stosie względem jego wierzchołka (wywołanie funkcji w Lua powoduje wrzucenie wszyst-

kich argumentów na stos w kolejności zgodnej z porządkiem parametrów funkcji). Zatem wywołanie funkcji postaci `Foo(n0, n1, op)` spowoduje sytuację na stosie zobrazowaną na Rysunku 1.

Po pobraniu argumentów ze stosu wykonujemy przykładowe operacje (dodawanie lub mnożenie) w zależności od trzeciego parametru przekazanego do funkcji. Wynik operacji wrzucamy na stos – w przypadku nierozpoznanej operacji wrzucamy wartość `nil`. Ostatnie polecenie `return 1` oznacza, iż zwracamy jedną wartość z naszej funkcji.

Mając zdefiniowaną funkcję na poziomie aplikacji, rejestrujemy ją poleceniem `lua_register(s, "Foo", FooFunc)` – oczywiście przed wykonaniem skryptu. Parametry `lua_register` oznaczają odpowiednio: stworzoną maszynę wirtualną Lua, odpowiednik nazwy funkcji w Lua oraz nazwę funkcji statycznej w C, którą rejestrujemy. Teraz już wszystko gotowe, aby odwołać się do naszej funkcji aplikacji z poziomu skryptu – Listing 8.

Znamy już sposób odwoływania się do funkcji aplikacji z poziomu skryptu Lua – teraz spróbujemy zrobić to samo w drugą stronę. Wcześniej w przykładach pokazaliśmy funkcję `factorial()` do obliczania silni. Zobaczmy, jak ponownie wykorzystać ideę stosowej maszyny wirtualnej i zawołać funkcję skryptową z poziomu kodu aplikacji (Listing 9).

Zaczynamy od wrzucenia na szczyt stosu funkcji `factorial()` z globalnej tablicy Lua. Jeżeli operacja powiodła się (wartość na szczycie stosu jest różna od nil), wrzucamy kolejno argumenty funkcji (w naszym przypadku jest to jedna liczba – w przykładzie jest to 4).

W ten sposób mamy przygotowane wywołanie funkcji, które wykonujemy poleceniem `lua_call`, podając liczbę parametrów wołanej funkcji (jeden) i liczbę zwracanych wartości przez funkcję (również jeden). Ostatnim krokiem jest pobranie ze szczytu stosu wyniku działania funkcji i wypisania go na ekran.

Praktyczne zastosowanie Lua

W poprzedniej części pokazaliśmy, w jaki sposób można zrealizować obustronną komunikację pomiędzy skryptem a kodem aplikacji. Spróbujmy teraz odnieść się do tych informacji poprzez praktyczne zastosowanie skryptu w konkretnym programie. Załóżmy, że tworzymy prostą grę typu arkanoid, gdzie zadaniem gracza jest odbijanie piłeczki paletką tak, aby zbijała one konkretne klocki ułożone na planszy. Chcemy umożliwić (spoza kodu aplikacji) pełną kontrolę zbijania klocków, która wiąże się z odgrywaniem różnych dźwięków, generowania efektów specjalnych czy przyznawania punktów graczowi w zależności od tego, w co uderzyła piłeczka. W tym celu przy każdej kolizji piłeczki z klockami na planszy będziemy wołać funkcję ze skryptu Lua.

W pierwszej kolejności musimy przygotować odpowiedni kod aplikacji, który obsługuje zderzenia piłeczki z innymi obiektami. Po wykryciu zderzenia będziemy wołać funkcję skryptu Lua, który podejmie odpowiednią akcję, bazując na informacji o zderzeniu. W naszym prostym przykładzie na tę informację będą składały się: nazwa typu klocka uderzonego przez piłeczkę i pozycja piłeczki na ekranie (Listing 10).

Nasza gra sprawdza, czy nastąpiło zderzenie piłeczki z klockiem na planszy (`IsCollision()`). Jeżeli tak, to wypełniana jest odpowiednia struktura z informacją o zderzeniu, a następnie wołana jest funkcja `OnCollision()` ze skryptu Lua. Należy oczywiście uprzednio przygotować argumenty tej funkcji – w naszym przypadku będzie to tablica z elementami opisującymi nazwę klocka i pozycję piłeczki na

Listing 7. *Przygotowanie funkcji aplikacji dla skryptu Lua*

```
static int FooFunc(lua_State *s)
{
   /// pobierz pierwszy argument
   float n0 = (float)lua_tonumber(s, -3);
   /// pobierz drugi argument
   float n1 = (float)lua_tonumber(s, -2);
   /// pobierz trzeci argument
   std::string op = lua_tostring(s, -1);

   float res = 0.0f;

   if (op == "add")
      res = n0 + n1;
   else if (op == "mul")
      res = n0 * n1;
   else
   {
      lua_pushnil(s);
      return 1;
   }

   /// wrzuć wynik na szczyt stosu
   lua_pushnumber(s, res);
   /// 1 oznacza liczbę zwracanych wartości
   return 1;
}
```

Listing 8. *Wywołanie funkcji aplikacji ze skryptu*

```
print(Foo(10, 14, "mul"));       --> 140
print(Foo(10, 14, "add"));       --> 24
print(Foo(10, 14, "div"));       --> nil
```

Listing 9. *Wywołanie funkcji skryptu z kodu aplikacji*

```
/// pobranie funkcji z globalnej tablicy skryptu
lua_getglobal(s, "factorial");
if (!lua_isnil(s, -1))
{
   /// wrzucenie na stos argumentu funkcji
   lua_pushnumber(s, 4);
   /// zawołanie funkcji
   lua_call(s, 1, 1);
   /// wyświetlenie wartości zwróconej
   std::cout << "Result from script function: " << lua_tonumber(s, -1) << std::
                     endl;
}
```

ekranie. Powyższy kod pokazuje, jak stworzyć i przekazać do maszyny wirtualnej tablicę postaci `event= {block = "name", x=0, y=0}` (oczywiście wypełnioną odpowiednimi wartościami). Przykładowa funkcja `OnCollision(event)` po stronie skryptu może wyglądać jak pokazano w Listingu 11.

Listing 10. *Obsługa kolizji w prostej grze typu arkanoid*

```
struct ScollInfo{
   /// nazwa typu klocka
   std::string Block;
   /// pozycja piłeczki w momencie zderzenia
   float   X, Y;
};

SCollInfo ci;

if (IsCollision(&ci))
{
   /// pobranie funkcji z globalnej tablicy skryptu
   lua_getglobal(s, "OnCollision");
   if (!lua_isnil(s, -1))
   {
      /// stworzenie nowej tablicy na szczycie stosu
      lua_newtable(s);
      /// zarejestrowanie table.block
      lua_pushstring(s, ci.Block.c_str());
      lua_setfield(s, -2, "block");
      /// zarejestrowanie table.x
      lua_pushnumber(s, ci.X);
      lua_setfield(s, -2, "x");
      /// zarejestrowanie table.y
      lua_pushnumber(s, ci.Y);
      lua_setfield(s, -2, "y");
      /// wywołanie funkcji skryptu
      lua_call(s, 1, 0);
   }
}
```

Listing 11. *Przykładowa funkcja w Lua obsługująca kolizje*

```
OnCollision = function(event)

   if event.block == "block_bonus" then
      PlaySound("bonus.wav");
      AddScore(100);
   elseif event.block == "block_death" then
      PlaySound("death.wav");
      AddParticles(event.x, event.y, "explosion");
      EndLevel(false);
   elseif event.block == "block_target" then
      PlaySound("fanfare.wav");
      AddParticles(event.x, event.y, "fanfare");
      AddScore(1000);
      EndLevel(true);
   elseif event.block == "block_normal" then
      local snd_table = {"hit_n1.wav", "hit_n2.wav", "hit_n3.wav"};
      PlaySound(snd_table[math.random(3)]);
      AddScore(10);
   else
      PlaySound("default.wav");
   end;

   Log(string.format("Collision with \"%s\" at (%f, %f)", event.block, event.x,
                     event.y));
end;
```

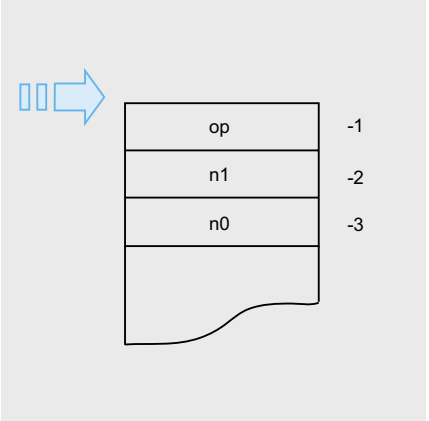
Funkcja sprawdza, z jakim typem klocka zderzyła się piłeczka i podejmuje odpowiednie czynności polegające na odgrywaniu odpowiedniego dźwięku, wyświetleniu efektów specjalnych czy zwiększeniu punktacji gracza. Oczywiście wszystkie funkcje aplikacji, czyli `PlaySounds`, `AddScore`, `AddParticles`, `EndLevel` czy `Log`, muszą być wcześniej zarejestrowane tak, aby Lua mogła je zawołać (tak jak opisaliśmy to wcześniej). I tak, kiedy piłeczka uderzy w klocek typu `block_death`, odgrywamy odpowiedni dźwięk, wyświetlamy odpowiedni efekt w miejscu piłeczki i przekazujemy do kodu aplikacji informację o niepomyślnym zakończeniu planszy. Natomiast w przypadku trafienia w klocek typu `block_normal` losujemy jeden z trzech dźwięków, odgrywamy go i dodajemy graczowi 10 punktów.

Dzięki takiemu podejściu możemy dodawać nowe typy klocków i kontrolować zachowanie podczas kolizji z nimi bez zmiany kodu aplikacji.

Standardowe biblioteki LUA

Lua posiada bogaty zestaw narzędzi wyższego poziomu zawartych we wbudowanych bibliotekach standardowych. Do tych bibliotek należą:

- operacje na łańcuchach znaków – narzędzia umożliwiające zaawansowane operacje na łańcuchach znaków, tj. wyszukiwanie, wyodrębnianie i zamiana podłańcuchów, dopasowywanie do wzorca, formatowanie tekstów itp.;
- operacje na tablicach – rozszerzenia umożliwiające bardziej wyrafinowane operacje na tablicach w Lua (np. sorto-



Rysunek 1. *Parametry funkcji na stosie oraz ich indeksy w Lua*

wanie elementów tablicy, konkatenacja tablic, wstawianie i usuwanie elementów);
- funkcje matematyczne – zbiór funkcji matematycznych, w tym funkcji trygonometrycznych, wykładniczych, wyznaczanie wartości bezwzględnej, minimalnej, maksymalnej, reszty z dzielenia, a także generowanie liczb losowych;
- funkcje wejścia/wyjścia – zbiór narzędzi operujących na plikach binarnych i tekstowych (tworzenie, otwieranie, zamykanie, czytanie z/pisanie do pliku),
- funkcje systemowe – biblioteka umożliwiająca podstawową interakcję z systemem operacyjnym (pobieranie daty, czasu, wykonywanie poleceń powłoki systemu);
- obsługa wątków – biblioteka obsługująca tworzenie, usuwanie i kontrolę wątków w Lua.

Przykład zamieszczony w Listingu 12 demonstruje użycie wybranych funkcji bibliotek standardowych Lua (generowanie liczb losowych, obliczenie sin(pi/2), sortowanie tablicy, formatowanie łańcucha znaków i pisanie do pliku tekstowego).

Dodatkowo Lua posiada wsparcie dla debugowania skryptów (*debug library*), tworzenia i zarządzania nowymi modułami (*modules*) oraz dodatkowych narzędzi ogólnego zastosowania, tj. iteratory tablic, konwertowanie typów danych czy możliwość łączenia kodu napisanego w kilku plikach.

Podsumowanie

W artykule zaprezentowaliśmy podstawy języka skryptowego Lua, a także sposób wykorzystania skryptów Lua w aplikacjach komputerowych tworzonych w C/C++. Lua dzięki swojej prostocie i przejrzystości umożliwia w łatwy sposób rozbudowę programów bez konieczności ponownej kompilacji ich kodu wynikowego. Takie podejście umożliwia sterowanie i wprowadzanie zmian do gotowego programu osobom niekoniecznie związanymi z programowaniem.

Niniejszy tekst wprowadza czytelnika w tematykę języka skryptowego Lua poprzez liczne przykłady praktycznego zastosowania.

Nie wyczerpuje to jednak tematu zaawansowanych mechanizmów jakimi są np. meta-tablice, będące specjalnymi atrybutami wszystkich typów danych w Lua. W skrócie, meta-tablice są (analogicznie do typów danych opisanych wcześniej) tablicami asosjacyjnymi definiującymi zachowanie poszczególnych zmiennych w stosunku do użytych operatorów arytmetycznych lub logicznych.

Taki mechanizm umożliwia nam wprowadzanie elementów programowania obiektowego poprzez możliwość przeciążania operatorów, a nawet korzystania z operacji dziedziczenia, czy polimorfizmu. Temat obiektowości w Lua ze względu na swoją obszerność nie został poruszony w tym artykule (zachęcam do odwiedzenia strony domowej Lua w celu uzyskania informacji na ten temat).

Lua cieszy się dużym powodzeniem nie tylko w branży gier komputerowych. Język ten zastosowano m.in. do opisania GUI w programie Adobe Photoshop Lightroom czy do elementów konfiguracji Apache HTTP server, a także do wielu innych zadań w komercyjnych i niekomercyjnych aplikacjach komputerowych.

Listing 12. *Przykład wykorzystania funkcji standardowych bibliotek Lua*

```
-- losowanie liczb
math.randomseed(os.time())
print(math.random(10));      -- wyświetlenie liczby wylosowanej z przedziału [1,10]

-- obliczenie wartości sin(pi/2)
print(math.sin(math.pi*0.5));      --> 1

-- sortowanie tablicy
a = { 10, 40, 20}
for i,n in ipairs(a) do print(n); end      --> 10 40 20
table.sort(a);
for i,n in ipairs(a) do print(n); end      --> 10 20 40

-- formatowanie łańcucha znaków
print(string.format("pi = %f", math.pi));      --> pi=3.141593

-- zapis do pliku
f = assert(io.open("output.txt", "w"))
f:write("test");
f:close();
```

PAWEŁ ROHLEDER

Autor pracuje na stanowisku programisty/projektanta grafiki komputerowej 3D we wrocławskiej firmie Techland Sp. z o.o. Jest również słuchaczem ostatniego roku studiów doktoranckich na Politechnice Wrocławskiej.
Kontakt z autorem: prohleder@wp.pl

W Sieci

- strona domowa LUA – *http://www.lua.org/*;
- podręcznik LUA on-line – *http://www.lua.org/manual/5.1/*;
- zbiór artykułów, dodatków i przykładów LUA – *http://lua-users.org/wiki/*.
- programowanie w Lua – *http://www.inf.puc-rio.br/~roberto/pil2/*.

LuaC

Lua umożliwia kompilację (za pomocą kompilatora LuaC) skryptów do postaci pośredniego kodu binarnego, który jest wykonywany na maszynie wirtualnej znacznie szybciej niż przebiegałaby jego interpretacja bezpośrednio z postaci tekstowej.

SQL Server 2008 Data Mining

Każdego dnia do firmowych baz danych napływają niezliczone ilości informacji. Dane składowane w tabelach zazwyczaj przekładają się na statystyki zakupionych towarów lub usług przez klientów. Ale coraz częściej posiadanie wiedzy, ilu konsumentów zostało obsłużonych w danym czasie, nie wystarcza.

Dowiesz się:
- W jaki sposób analizować dane za pomocą Data Mining;
- O dostępnych mechanizmach DM w SQL Server 2008;
- Jak wykonać prosty przykład analizy danych.

Powinieneś wiedzieć:
- Podstawy pracy z BIDS;
- Podstawowe pojęcia statystyki matematycznej i probabilistyki;
- Znajomość środowiska SQL Server oraz Analysis Services.

Poziom trudności

Dobry manager chce wiedzieć więcej chociażby o profilu swoich nabywców, aby móc zaprojektować skuteczną kampanię reklamową. Dlatego istnieje potrzeba formowania tychże danych w użyteczne informacje oraz w rezultacie – w wiedzę.

Niniejszy artykuł poruszać będzie tematykę wykorzystania zaawansowanych mechanizmów Data Mining z Microsoft Business Intelligence Development Studio oraz SQL Server 2008. Zostanie omówiony algorytm naiwny Bayesa oraz jego użycie w kontekście tworzenia idealnego profilu klienta firmy AdventureWorksDW (Data Warehouse). Zaprezentuję również technikę klastrowania (clustering) w procesie formowania obszarów (skupisk)potencjalnych nabywców oraz ich profili.

Techniki eksploracji danych

Eksploracja danych (Data Mining) jest jednym z mechanizmów wydobywania wiedzy z bazy danych, wywodząca się głównie z obszarów badań nad sztuczną inteligencją. Mechanizmy te można zgrupować w następujących zakresach:

- *Logika rozmyta* – uogólnienie logiki binarnej – w obszarze między 0 a 1 znajdują się wartości pośrednie, określające przynależność elementu do danego zbioru;
- *Sieci neuronowe* – przetwarzanie sygnałów przez sieć sztucznych neuronów, wykonujących określoną operację na wejściu;
- *Metody ewolucyjne* – poszukiwanie najlepszego rozwiązania z całej przestrzeni dostępnych rozwiązań;
- *Metody statyczne* – wykonanie obliczeń z zakresu statystyki matematycznej (np. średniej arytmetycznej, odchylenia standardowego itp.) dla określonej grupy reprezentacyjnej.

Większość analityków myli proces eksploracji danych z procesem odkrywania wiedzy z bazy (ang. *Knowledge Discovery in Databases (KDD)*). Otóż na KDD składa się wiele czynników, z czego Data Mining wykorzystuje się w piątym kroku, zgodnie z poniższym schematem:

- Czyszczenie (*Data Cleaning*) – usuwanie duplikatów i niespójności;
- Integracja (*Data Integration*) – łączenie danych z wielu źródeł;
- Wybór (*Data Selection*) – pobranie danych, które będą podlegały analizie;
- Transformacja (*Data Transformation*) – przekształcanie, wykonywanie skróconej agregacji;
- Eksploracja danych (*Data Mining*) – stosowanie inteligentnych metod w celu wyodrębniania wzorców;
- Schemat oceny (*Pattern Evaluation*) – analiza otrzymanych wyników;
- Prezentacja wiedzy (*Knowledge Presentation*) – wykorzystanie dostępnych mechanizmów prezentacji (histogramy, wykresy kołowe, tabele) do zobrazowania danych.

Szczegółowa architektura systemu opartego o Knowledge Discovery in Databases została zaprezentowana na Rysunku 1.

Ponieważ proces wydobywania wiedzy w systemach KDD jest niezwykle złożony, w niniejszym artykule pragnę skupić się jedynie na częściach związanych z eksploracją danych oraz schematami oceny wyników.

Algorytm naiwny Bayesa i drzewa decyzyjne

Istnieją dwa sposoby analizy danych, które mogą być wykorzystane do wyodrębniania modeli opisujących ważne informacje: klasyfikacja (ang. *classification*) oraz predykcja (ang. *predication*). Klasyfikacja polega na sposobie przydzielania atrybutów bazujących na obserwacjach do zbioru predefiniowanych klas. Natomiast predykcja jest mechanizmem przewidywania procesu kształtowania się zdarzeń w przyszłości. Przykładowo, w kontekście bazy AdventureWorksDW można wyodrębnić reguły klasyfikujące czynniki zakupu ilości rowerów (miejsce zamieszkania, stan cywilny, odległość od miejsca pracy), a następnie w odniesieniu do tej wiedzy przypisać klientów przedstawiających się odpowiednimi cechami. W przeciwieństwie do klasyfikacji, predykcja wspomoże określić wpływ czynników wewnętrznych oraz zewnętrznych na kształtowanie się trendu w zakupie danej marki roweru w przyszłości.

Przykładem statycznej klasyfikacji probabilistycznej jest *Naiwny Klasyfikator Bayesa*, oparty na założeniu o niezależności predykatów. Metoda ta jest często wykorzystywana w sztucznej inteligencji, gdyż wspomaga podejmowanie de-

cyzji na podstawie złożonej analizy warunkowego prawdopodobieństwa wystąpienia zdarzeń oraz uczenia maszynowego. Podstawowym wynikiem klasyfikacji jest dodanie przypadku do jednej z kategorii decyzyjnych, wyliczonego ze wzoru Twierdzenia Bayesa:

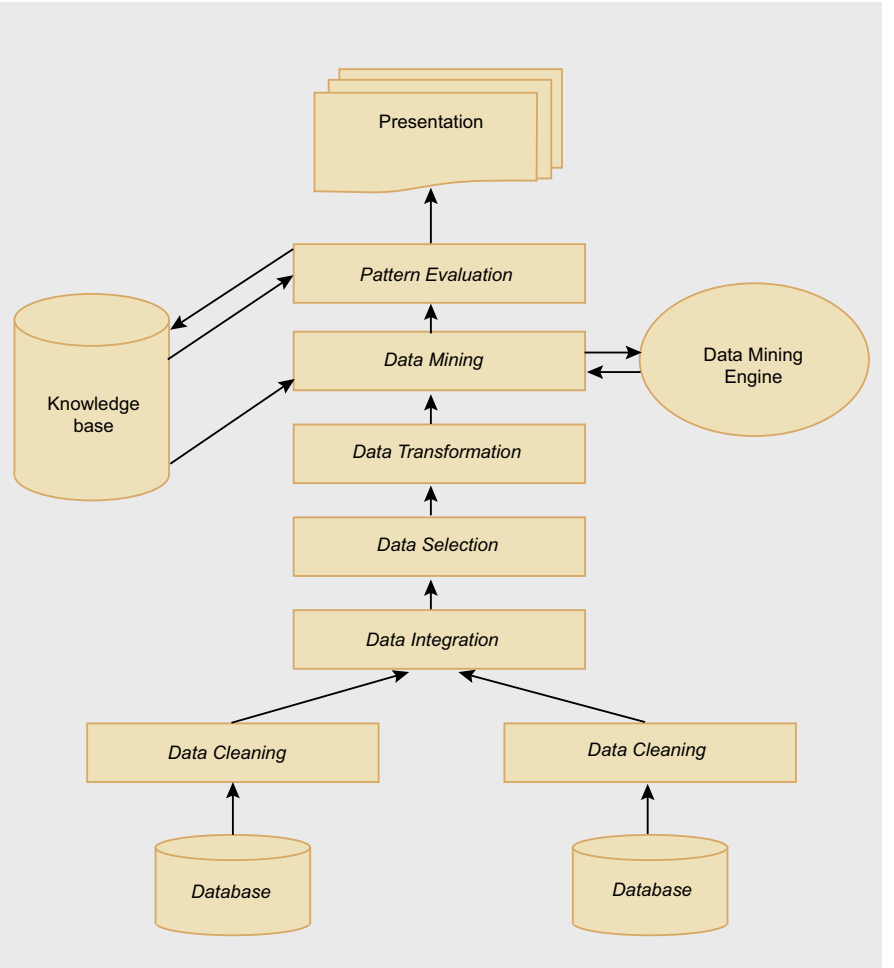
```
PAiB= PBAiP(Ai)j=1NPBAjP(Aj)
```

gdzie `A1… An` są zdarzeniami niezależnymi, natomiast `PA1…PAn` prawdopodobieństwami wystąpnienia zdarzeń. Algorytm Naiwny Bayesa często jest wykorzystywany do wspomagania konstrukcji drzew decyzyjnych, czyli graficznej metody wspomagania procesu podejmowania decyzji. Z punktu widzenia informatyki, *Decision Tree* jest grafem, którego korzeń tworzony jest przez wybrany atrybut, natomiast poszczególne gałęzie stanowią prezentację jego wartości.

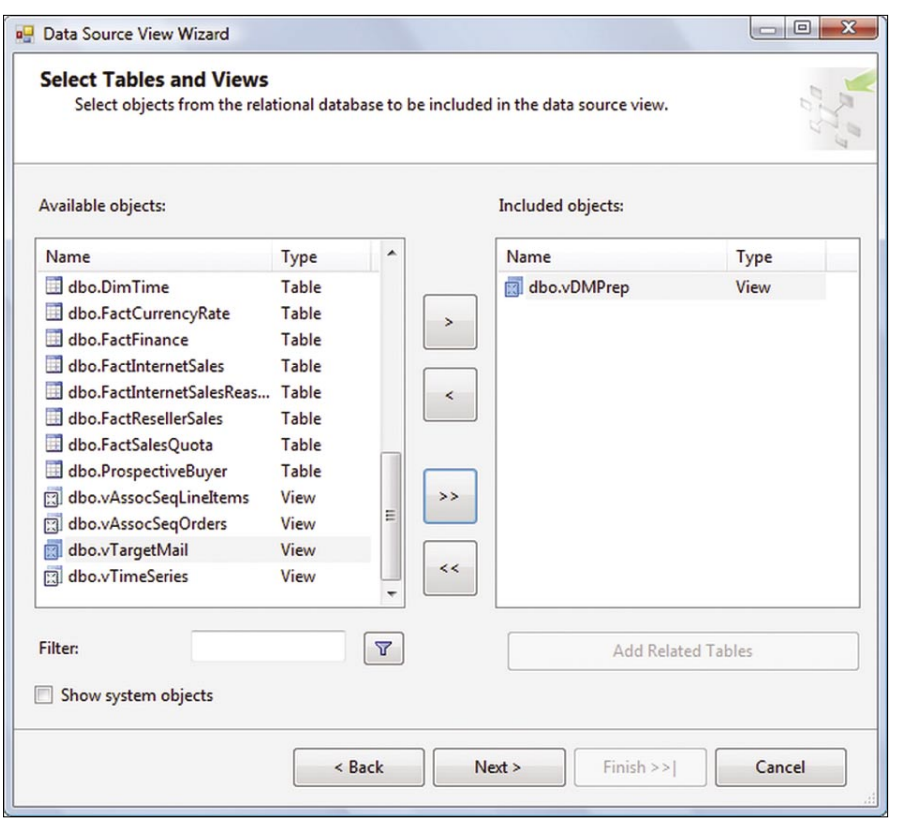
Przykład I – profil idealnego klienta

Po krótkim wstępie teoretycznym chciałbym zaprezentować czytelnikowi sposoby wykorzystania algorytmów Data Mining w Business Intelligence Development Studio oraz SQL Server 2008. Jako przykład wybrałem utworzenie profilu idealnego klienta na podstawie danych zgromadzonych w bazie AdventureWorksDW. W pierwszym kroku w Visual Studio (BIDS) należy wybrać nowy projekt Business Intelligence *Projects -> Analysis Services Project*. W rezultacie zostanie utworzona solucja składająca się z następujących części:

- *Data Sources* – definicja źródeł danych;
- *Data Sources Views* – widoki, tabele, z których będą pobierane dane;
- *Cubes* – źródła danych pochodzących z kostek OLAP;
- *Dimensions* – wymiarowość w kostkach OLAP;
- *Mining Structures* – definicje struktur Data Miningowych operujących na zbiorze danych;
- *Roles* – utworzenie ról i praw dostępu – zarządzanie poziomem dostępności;
- *Assemblies* – referencje do komponentów zewnętrznych;
- *Miscellaneous* – folder na pozostałe pliki, dokumenty.



Rysunek 1. *Architektura systemu Knowledge Discovery in Database*



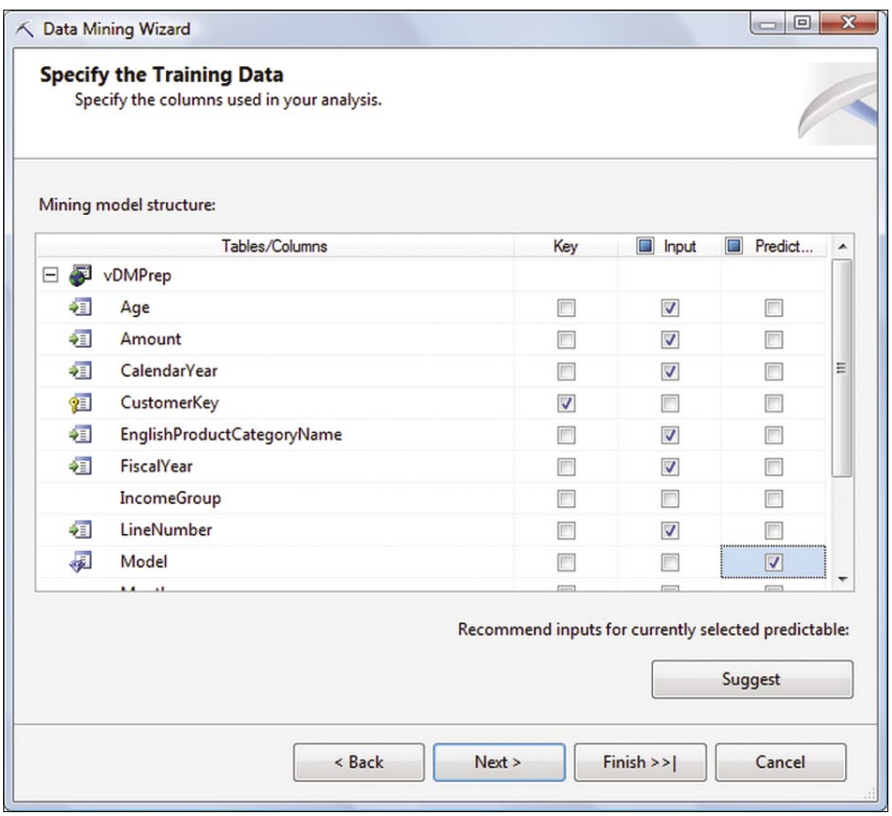
Rysunek 2. *Wybór widoku vDMPrep*

Bibliografia

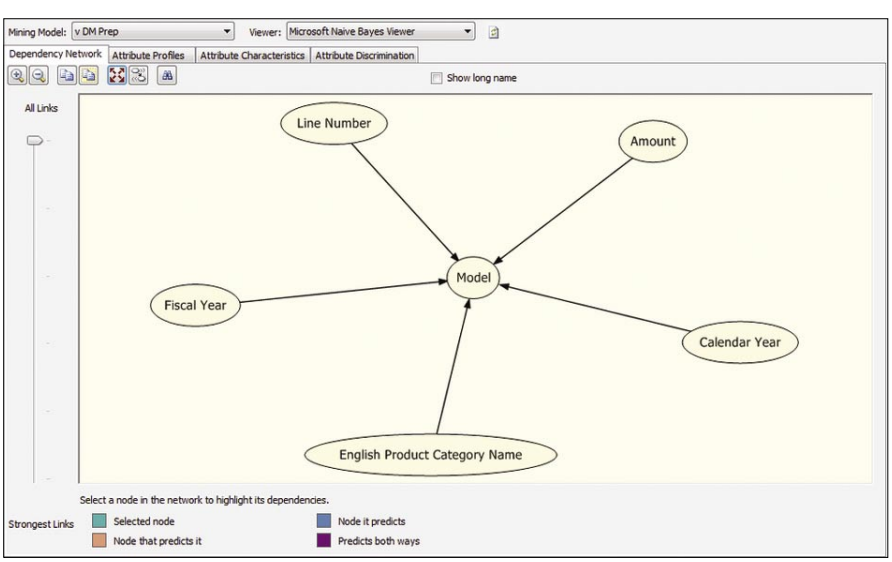
- J. MacLennan, *Data Mining with SQL Server 2008*, Wiley Publishing, 2008;
- C.L. Curotto, *Implementing Data Mining Algorithms in Microsoft SQL Server 2008*, WTI Press, 2005;
- B. Larson, *Delivering Business Intelligence with Microsoft SQL Server 2008*, McGraw-Hill Company, 2009.

W kolejnym kroku należy zdefiniować źródło danych. Aby to uczynić, wystarczy kliknąć prawym klawiszem myszy na folderze *Data Sources* i wybrać *New Data Sources*. Po poprawnym określeniu serwera bazodanowego oraz samej bazy, z której będą pobierane informacje, trzeba zawęzić obszar eksploracji danych do tabel bądź widoków. Z menu podręcznego *Data Sources Views* należy wybrać opcję *New Data Sources Views*. Spowoduje to uruchomienie kreatora, który ułatwi proces selekcji odpowiednich tabel. Na potrzeby edukacji w bazie *AdventureWorksDW* został utworzony widok vDMPrep, którego wybiera się w sekcji *Select Tables and Views*, klikając przycisk *Add*(Rysunek 2). Po utworzeniu widoków przychodzi kolej na wybór mechanizmów, które będą użyte do eksploracji danych. Podobnie jak w poprzednich krokach, wystarczy z menu podręcznego na katalogu *Mining Structures* wybrać opcję *New Mining Structure*. Ponieważ analiza danych będzie budowana w oparciu o Algorytm Naiwny Bayesa, w kroku *Select the Data Mining Structures* wartość *Data Mining Technique* powinna być ustawiona na *Microsoft Naive Bayes*. Przechodząc do kolejnych etapów kreatora, chciałbym na chwilę zatrzymać się w *Specify The Training Data*. Na tym poziomie użytkownik decyduje, jakie wartości będą składały się na klasyfikację, a jakie na predykcję. W niniejszym przykładzie, znając cechy klienta *AdventureWorks*, będziemy chcieli ustalić regułę przyszłego wnioskowania dla zakupu roweru. Zatem tabelę tę należy wypełnić według następujących zaleceń (Rysunek 3):

- *Key*: *Customer Key*;
- *Input: Age, Amount, CalendarYear, Region, FiscalYear, LineNumber, EnglishProductCategoryName*;
- *Predictable: Model*.



Rysunek 3. *Określenie wartości Key, Input, Predictable*



Rysunek 4. *Sieć zależności*

Zaznaczając wyłącznie wartość klucza głównego oraz predykcji, można skorzystać z opcji autosugestii (przycisk *Suggest* – Rysunek 3). Spowoduje to utworzenie listy zalecanych wartości wejściowych dla danej reguły wnioskowania. Krok *Create Testing Set* jest nowością w SQL Server 2008 i w przeciwieństwie do swojego poprzednika pozwala na określenie mnogości zbioru danych przeznaczonych do prób (testów). Można go ustalić procentowo bądź jako generalne ograniczenie. Domyślnie 30% danych wykorzystuje się do testów i 70% do właściwej analizy, bez jednoznacznego typowania maksymalnej liczby przypadków testowych.

Po zakończeniu pracy kreatora model jest gotowy do utworzenia oraz wdrożenia w środowisku SQL Server. Aby to uczynić, należy wybrać przycisk *Create a related Mining Model*, znajdujący się w Menu zakładki *Mining Structure*. Algorytm operujący na przygotowanej strukturze to *Microsoft Decision Tree*, zatem taki powinien zostać wybrany z listy dostępnych algorytmów. Ostatni krok to wdrożenie projektu. Podobnie jak w przypadku solucji .NET-owych z BIDS, należy wybrać opcję *Build->Deploy [Nazwa Projektu]*. W tym momencie przygotowana solucja została osadzona w zasobach SQL Server 2008. Jej fizyczną implementację można podglądnąć, łącząc się z serwerem Analysis Services np. poprzez Management Studio. Struktura katalogów w Object Explorerze powinna się zga-

W Sieci

- Webcast-y prezentujące wprowadzenie do tematyki Data Mining w środowisku Microsoft SQL Server 2008: *http://www.microsoft.com/sql/2008/datamining/index.html*;
- Books Online: Analysis Services oraz Business Intelligence Development Studio: *http://technet.microsoft.com/en-us/library/ms173709.aspx*;
- Obszerne wprowadzenie do tematyki Data Mining: *http://www-users.cs.umn.edu/~kumar/dmbook/index.php*;
- Wprowadzenie do Business Intelligence Development Studio: *http://msdn.microsoft.com/en-us/library/ms173767.aspx* .

dzać z przygotowaną w BIDS. Po poprawnym wdrożeniu projektu można przejść do zakładki *MiningModel Viewer* oraz korzystać z pełnego dobrodziejstwa analizy danych, jaką dostarcza nam SQL Server 2008. Wybierając z listy rozwijanej *Mining Model -> vDMPrep*, otrzymamy sieć zależności poszczególnych wartości wejściowych od wyboru modelu (Rysunek 4). Przesuwając suwak po lewej stronie, można się zorientować, który z nich jest najistotniejszy.

Najbardziej interesujące będzie jednak przeglądanie drzewa decyzyjnego. Jego graficzną reprezentację można otrzymać, zmieniając Mining Model z vDMPrep na nazwę modelu, który został utworzony w poprzednim punkcie. W rezultacie wygenerowany zostanie graf, zgodnie z Rysunkiem 5.

Przykładowa analiza do wygenerowanego przypadku: Jeżeli klient zarabia mniej niż 440.392, szansa, że zakupi rower marki WaterBottle to ok 21%. Dla linii < 3 sytuacja się zmienia, gdyż prawdopodobieństwo zakupu tegoż modelu spada o 5 %. Okazuje się również, iż fakt zamieszkiwania Północnej Ameryki ma istotny wpływ na wybór WaterBottle. Dla mieszkańca i obcokrajowca prawdopodobieństwo zakupu wynosi odpowiednio 17 i 14%. Posiadając informacje o zarobkach klienta wynoszących powyżej 2641.376 oraz obywatelstwa amerykańskiego, może sugerować sprzedawcy, iż Road-150 będzie na pewno najchętniej wybieranym (prawdopodobieństwo ok. 98%) rowerem. Zatem oddział sklepu w dzielnicy, których profile klientów zbliżone są do wcześniej wymienionych danych, powinien zostać niezwłocznie wyposażony w sprzęt odpowiednich marek.

Przykład 2 – Clustering

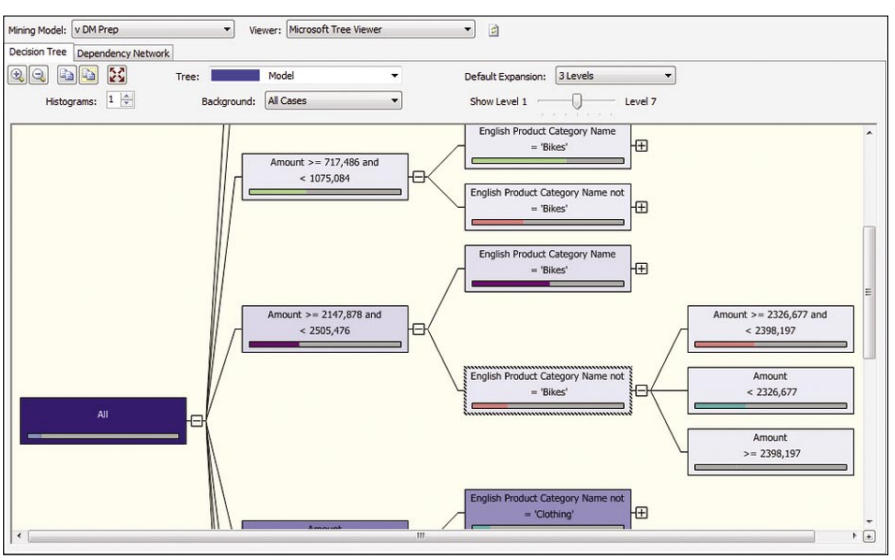
W najprostszym tłumaczeniu *Clustering* polega na efektywnym tworzeniu sensownych grup powiązanych tematycznie. Algorytm tworzenia skupisk składa się z następujących kroków:

- wyodrębnienie grup tematycznych;
- znalezienie przewodniej cechy wspólnej dla każdej z grup;
- przypisanie wszystkich pozycji do danych grup.

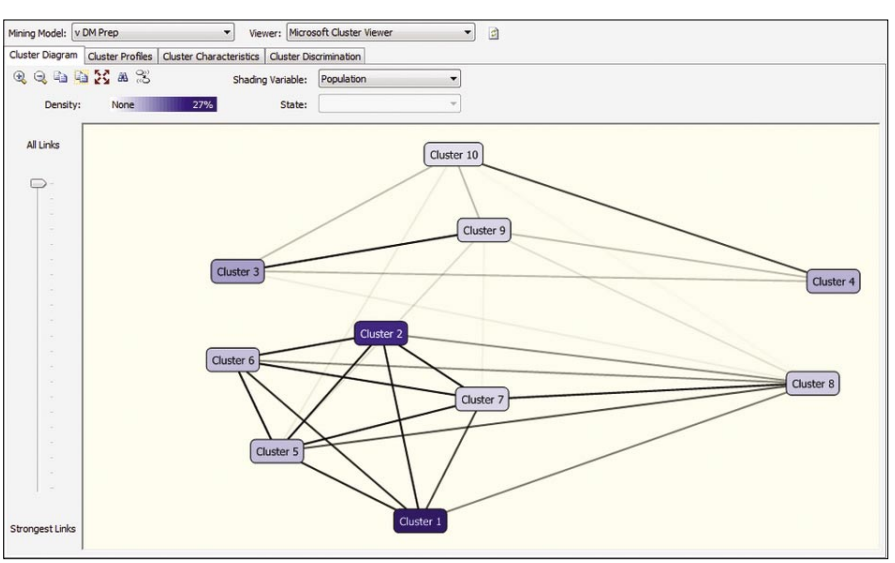
Zastosowanie algorytmu klastrowania można odnaleźć niemal wszędzie, np. w biologii: klasyfikacja roślin i zwierząt ze względu na funkcje, jakie pełnią, bądź w branży ubezpieczeniowej: identyfikacja kierowców i ich pojazdów z wysoką średnią roszczeń polis ubezpieczeniowych. W naszym przypadku chciałbym dokonać klasyfikacji klientów według cech ustalonych w poprzednim punkcie. Każda z grup pomoże odpowiedzieć na pytanie, jaki rower jest najchętniej wybierany w danym środowisku.

Narzędzia Business Intelligence posiadają bardzo wygodną cechę – mianowicie do raz stworzonej struktury Data Miningowej można utworzyć wiele modeli. Aby tego dokonać, należy ponownie wybrać przycisk *Create a related mining model*, tym razem jednak jako *Algorithm name* wybrać *Microsoft Clustering*. Po poprawnym wdrożeniu projektu do środowiska SQL Server można przeglądać analizę, podobnie jak na Rysunku 6.

Analiza przykładowego skupiska: W obszarze 1 (Cluster 1) zamieszkują ludzie wyłącznie w wieku 39 – 47 lat. Są w głównej mierze Amerykanami bądź Europejczykami. Ich zarobki nie przekraczają 440.39215047. Rok kalendarzowy w większości przypadków to 2003 oraz 2004. Modele rowerów, jakie kupują, są raczej niestandardowe (poza katalogiem), ale odnotowano również sprzedarz WaterBottle, Patch kit, Mountain Tire T.



Rysunek 5. *Drzewo decyzyjne – klient AdventureWorksDW*



Rysunek 6. *Sieć Microsoft Clustering*

Podsumowanie

Data Mining obecnie przeżywa swój informatyczny renesans. Przez wiele lat ta dziedzina wiedzy była postrzegana raczej jako zawiła i nie do końca rozumiana ciekawostka, pozbawiona praktycznego zastosowania. W dobie masowych informacji, chaosu danych oraz przetwarzania rozproszonego te, wbrew stereotypom, relatywnie proste algorytmy znalazły zastosowanie w takich dziedzinach życia publicznego jak banki, giełdy papierów wartościowych, logistyka, sprzedarz – wspomagając symulację analiz finansowych, operacji pieniężnych czy prowadzenia zwyczajnej kampanii reklamowej.

PAWEŁ WILKOSZ

Paweł Wilkosz swoje zainteresowania koncentruje wokół rozwiązań firmy Microsoft (SQL Server, .NET) oraz Business Intelligence. Jest aktywnym członkiem Polish SQL Server User Group w Krakowie, uczestnicząc w spotkaniach jako członek oraz prelegent. Na codzień pracuje jako Software Quality Engineer w polskim oddziale firmy Making Waves. Kontakt z autorem: pawel.wilkosz@gmail.com.

Dogadajmy się

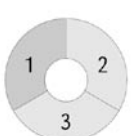
O komunikacji dla programistów

Im więcej osób pracuje nad projektem, tym bardziej powodzenie przedsięwzięcia zależy od efektywnej komunikacji pomiędzy nimi. Prace nad projektami IT dotyczą programistów, architektów, analityków. Systemy są udoskonalane przez lata, a zaangażowane osoby zmieniają się. W artykule przyglądamy się aspektom komunikacji w zespołach.

Dowiesz się:
- Jak dbać o efektywną komunikację w trakcie prac nad projektem;
- Jakie konkretne czynniki mogą polepszać lub pogarszać komunikację w zespole projektowym.

Powinieneś wiedzieć:
- Tylko to, co już wiesz.

Poziom trudności



Potoczne rozumienie słowa komunikacja nasuwa skojarzenia związane z pracą z klientem, z kompetencjami w dziedzinie zarządzania czy z pracą w grupie. Wszak handlowiec musi się sprawie komunikować ze swoim klientami, dobry menedżer również powinien dbać o jakość komunikacji ze swoimi podwładnymi. A co z programistami? Czy zdolność komunikacji również wpływa na jakość ich pracy? Czy należy na nowo zdefiniować pojęcie *komunikacja* w odniesieniu do programistów zespołów programistycznych?

Artefakty

Myśląc o komunikacji, zazwyczaj mamy na myśli komunikację międzyludzką. Tymczasem jest to tylko jeden aspekt komunikacji pomiędzy osobami biorącymi udział w projekcie. Bardzo wiele istotnych informacji programiści przekazują sobie poprzez kod źródłowy oraz poprzez różnego rodzaju dokumenty formalne i nieformalne. Zapożyczając terminologię z metodyki RUP, ten sposób komunikacji nazwiemy komunikacją poprzez artefakty.

Kod źródłowy

Zdecydowanie najwięcej informacji programiści przekazują sobie poprzez kod źródłowy. To za pomocą języków programowania wyrażają swoje myśli i wcielają w życie pomysły na zaimplementowanie nowych funkcjonalności systemu. Podobnie jak w języku naturalnym, funkcja komunikacyjna języka programowania polega na przekazaniu odbiorcy komunikatu zgodnego z intencją nadawcy. Odnosząc się do tej analogii, zastanówmy się, jakie są składowe efektywnej komunikacji pomiędzy programistami za pomocą języka programowania.

Podstawowym celem jest przekazywanie informacji w sposób zrozumiały dla odbiorcy. Czasy, w których napisanie kodu za pomocą jak najmniejszej ilości znaków, wykorzystując przy tym niezliczoną ilość efektów ubocznych, bezpowrotnie minęły. W chwili obecnej kod źródłowy jest częściej czytany, niż pisany, a łatwość zrozumienia i rozbudowywania przez programistę istniejącego kodu znacząco wpływa na efektywność jego pracy. Z potrzeby czytelności wynikają koncepcje takie jak *samodokumentujący się kod* czy *czytaj kod jak książkę*.

Dokumenty

Drugim aspektem komunikacji w zespole projektowym są różnego rodzaju dokumenty formalne lub nieformalne dotyczące projektu i opisujące poszczególne rozwiązania. Niektóre z nich są ściśle określone przez procedury, według których działa zespół. Inne są inwencją twórczą samych programistów. Najpopularniejszym narzędziem do przygotowywania dokumentacji pro-

Dbaj o kod źródłowy

- stosuj standard kodowania;
- używaj opisowych nazw zmiennych, metod i klas;
- precyzyjnie rozdzielaj odpowiedzialność pomiędzy klasy, metody oraz zmienne;
- korzystaj z wzorców projektowych;
- rozważania nad czytelnością kodu źródłowego znajdziesz w książkach K.Becka *Implementation Patterns* oraz R.C.Martina *Clean Code*.

Dbaj o diagramy UML

- na diagramie przekazuj tylko jeden aspekt funkcjonowania systemu, unikaj dużej ilości nieistotnych szczegółów;
- duże i złożone diagramy podziel na kilka mniejszych obejmujących węższe obszary działania aplikacji;
- najistotniejsze elementy umieszczaj w centrum diagramu;
- jeśli posługujesz się diagramami, to dbaj, aby zawsze odzwierciedlały aktualny stan kodu;
- utrzymuj porządek w swoich diagramach.

jektowej są diagramy UML. W założeniu ten Ujednolicony Język Modelowania (ang. *Unified Modeling Language)* ma dostarczać medium komunikacji pomiędzy analitykami, projektantami, architektami, programistami, testerami. Obowiązująca specyfikacja UML definiuje kilkanaście różnych diagramów. Problem w tym, że choć sam język jest precyzyjnie określony, to pomysłów na jego używanie w trakcie wytwarzania oprogramowania jest bardzo wiele. Innymi słowy użytkownik UML staje przed dylematami: których diagramów powinien używać, kiedy ich używać, czy istnieje minimalny podzbiór UML wystarczający dla danego projektu, jak efektywnie korzystać z UML? Postawione pytania sprowadzają się do zasadniczego problemu: jak wygląda optymalny proces pracy z UML w moim projekcie? W tym temacie powstało bardzo wiele różnych pomysłów, najczęściej zdefiniowanych poprzez metodyki wytwarzania oprogramowania lub sugerowane poprzez narzędzia wspomagające modelowanie. Tymczasem w wielu przypadkach programiści używają diagramów w razie potrzeby w sposób, który podpowiada im intuicja. Dopóki powstałe w ten sposób modele wykorzystywane są jedynie przez programistów (w obrębie jednego zespołu), spełniają one swoją funkcję komunikacyjną. W momencie, gdy w cykl pracy wchodzą analitycy, architekci lub podwykonawcy czy partnerzy biznesowi, sprawy zaczynają się komplikować. Wtedy brak dobrze zdefiniowanego procesu z udziałem UML wprowadza bardzo wiele komplikacji, a same diagramy zamiast ułatwiać komunikację, utrudniają ją.

Tropem pewnego stereotypu

Jeden z naszych przyjaciół powiedział kiedyś, że programista kojarzy mu się z kimś mało komunikatywnym, kto cały czas spędza przed komputerem i wstaje sprzed niego tylko wtedy, gdy mu się podkoszulek złuszczy... Nie było to zbyt przyjemne porównanie, ale oddawało część tego, w jaki sposób nieprogramiści nas postrzegają. Przeanalizujmy poniższy przykład, aby zbadać źródło tego stereotypu.

Z perspektywy nieprogramisty

Jacek (l.31), psycholog: *Dziwnie się czuję, gdy wchodzę do pokoju informatyków. Nikt mnie tam nie zauważa. Wszyscy siedzą wpatrzeni w monitory i coś tam piszą. Gdy zapytałem, kiedy CRM będzie działał, powiedzieli mi: „w logach jest, że dżara nie można zdeplojować". Jakby to cokolwiek dla mnie znaczyło...nowomowa jakaś. Ostatnio mieliśmy spotkanie na temat nowej wersji naszego narzędzia. Chciałem wytłumaczyć, jak to ma działać, nawet trochę rysowałem, żeby być dokładnym. Oni cały czas drążyli szczegóły, jakby w ogóle nie interesowało ich, jak ten system ma działać. Zadawali mi mnóstwo pytań, na które nie potrafiłem odpowiedzieć. Przecież nie wiem, jak dokładnie to narzędzie ma wyglądać, bo jeszcze takiego nie ma na świecie. Po spotkaniu dostałem mailem podsumowanie i zbladłem. Wydawało mi się, że rozmawialiśmy i ustaliliśmy zupełnie coś innego.*

Z perspektywy programisty

Tomek (l.27), Senior Java Developer: *Wczoraj zbieraliśmy wymagania do wersji 1.3. Właściwie to nie wiem, po co odbyło się to spotkanie. Jacek kompletnie nie miał pojęcia, czego chce. Pytaliśmy, jak mają działać poszczególne algorytmy, w jaki sposób wyobraża sobie prace z systemem, i nie mogliśmy nic z niego wydusić. Przytaczał mnóstwo ogólnych przykładów, ale nic konkretnego nie powiedział. Gdy próbowaliśmy podsumować i sprecyzować jego oczekiwania, to odpowiadał „tak, ale...". Żeby dodać te funkcjonalności, potrzebujemy dobrze określonych reguł ich działania, a nie garści przypadków szczególnych. I w dodatku on ciągle nazywa nas informatykami, a przecież programista i informatyk to dwie kompletnie różne rzeczy.*

Na zewnątrz zespołu

Jacek i Tomek mieli wspólny cel – dodać nowe funkcjonalności do narzędzia wykorzystywanego w ich firmie. To, czy uda im się go osiągnąć, zależy przede wszystkim od tego, na ile będą potrafili efektywnie ze sobą współpracować.

Jacek wiedział, jaki efekt chce uzyskać. Koncentrował się na ostatecznej wizji systemu. Nie zastanawiał się nad tym, *jak* stworzyć system, lecz *co* on ma robić i jakie funkcjonalności dostarczać. W jego opinii szczegółami technicznymi powinni zająć się programiści.

Tomek chciał napisać aplikację, o której mówił Jacek. W dodatku był doświadczonym programistą i wiedział, że trzeba jednoznacznie określić wymagania klienta, aby dostarczyć mu taki system, jakiego rzeczywiście chce. Ponieważ Jacek nie potrafił odpowiedzieć na konkretne pytania, programista uznał, że użytkownik nie przemyślał swoich oczekiwań.

Programowanie to wyjątkowo złożone zajęcie, wymaga dużego skupienia, dokładności i panowania nad ogromną ilością szczegółów. Błędy mogą drogo kosztować. Dlatego, jako programiści, wypracowaliśmy sobie umiejętności, które pomagają nam efektywnie wykonywać pracę. Wśród najbardziej cennych można wymienić umiejętności:

- skupienia się na szczegółach;
- dekomponowania problemu na elementarne składowe;
- odnajdywania różnic i podobieństw, co pozwala na szybkie lokalizowanie błędów;
- analizy i syntezy, dzięki którym projektujemy algorytmy.

Umiejętności te doskonale się sprawdzają w pracy nad projektem, ale w innym środowisku, np. podczas rozmowy z potencjalnym użytkownikiem, *Product Managerem* czy innym programistą, potrzeba odmiennych umiejętności. Przede wszystkim słuchania i rozumienia potrzeb. Duży krok w stronę rozwoju kompetencji komunikacyjnych poczyniły metodyki z nurtu *Agile*, postulując w swym manifeście *ludzie i interakcje ponad procesy i narzędzia* oraz *współpraca z klientem ponad negocjowanie kontraktów.* Przejawem tych paradygmatów jest praktyka *planning game* z metodyki eXtreme Programming. Mimo tego, potrzeba rozwoju umiejętności komunikacyjnych wciąż bardzo wolno przedziera się do świadomości programistów.

Tabela 1. *Podobieństwa pomiędzy językiem naturalnym a językiem programowania w odniesieniu do ich funkcji komunikacyjnej*

Język naturalny	Język programowania
Wyrazy, zdania, wypowiedzi pisane	Słowa kluczowe, procedury, klasy, metody, zmienne, kod źródłowy
Styl języka	Standard kodowania
Frazeologia	Wzorce projektowe
Jednoznaczność komunikatu, unikanie generalizacji, koncentrowanie się na faktach	Dbanie o jednoznaczne wydzielenie odpowiedzialności dla zmiennych, metod, klas, pakietów i modułów

MICHAŁ BARTYZEL, MARIUSZ SIERACZKIEWICZ

Trenerzy i konsultanci w firmie BNS IT. Badają i rozwijają metody psychologii programowania, pomagające programistom lepiej wykonywać ich pracę. Na co dzień Autorzy zajmują się zwiększaniem efektywności programistów poprzez szkolenia, warsztaty oraz coaching i trening.

Kontakt z autorami: m.bartyzel@bnsit.pl, m.sieraczkiewicz@bnsit.pl

Analiza biznesowa

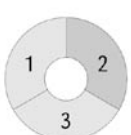
Wymagania – pozyskiwanie, dokumentowanie, komunikowanie, weryfikacja.

Etap analizy biznesowej jest jednym z kluczowych momentów w projekcie informatycznym. Jakość analizy i uzyskanych za jej pomocą produktów w znacznym stopniu determinuje jakość kolejnych etapów realizacji systemu. Niniejszy artykuł przedstawia podstawowe elementy dobrego procesu analizy.

Dowiesz się:
- Jakie czynności wchodzą w skład analizy biznesowej;
- Jakie czynniki wpływają na jakość produktów analizy.

Powinieneś wiedzieć:
- Średnio zaawansowana wiedza ogólna z zakresu analizy biznesowej i modelowania wymagań;
- Podstawowa wiedza dotycząca prowadzenia projektów informatycznych.

Poziom trudności



Zadaniem analityka biznesowego jest, najkrócej mówiąc, pozyskanie wymagań dotyczących oczekiwań klienta wobec planowanego systemu oraz takie zamodelowanie i udokumentowanie owych wymagań, by umożliwiały sprawną i poprawną implementację aplikacji. Analityk jest ponadto odpowiedzialny za komunikowanie wymagań członkom zespołu i udzielanie wsparcia merytorycznego wszędzie tam, gdzie jest ono potrzebne – podczas tworzenia kodu czy testowania.

Nie da się przecenić roli analizy biznesowej – szczególnie w przypadku, gdy tworzone jest nowe rozwiązanie (całkowicie nowy system). Samo spisanie wymagań nie wystarcza – owszem, daje wgląd w przybliżony zakres oczekiwań klienta i funkcje, jakie ma dostarczać dany system, ale nie obejmuje zależności pomiędzy poszczególnymi funkcjonalnościami, modułami systemu, nie określa możliwych interakcji z użytkownikiem, ewentualnych wyjątków etc.

Czym jest analiza biznesowa?

Najogólniej rzecz ujmując, analiza biznesowa jest procesem, którego celem jest zidentyfikowanie potrzeb klienta w zakresie funkcjonalności oraz innych biznesowych aspektów dotyczących systemu informatycznego.

Zgodnie z wytycznymi BABOK w skład analizy biznesowej wchodzą następujące elementy (Rysunek 1):

- Analiza przedsiębiorstwa, która polega na identyfikacji i zrozumieniu potrzeb danego biznesu (branży). Jest to etap poprzedzający właściwe pozyskiwanie wymagań, umożliwia zapoznanie się z celami oraz charakterystyką danej organizacji celem lepszego zorganizowania procesu analizy biznesowej.
- Pozyskiwanie wymagań – czyli proces identyfikacji oczekiwań klienta, zarówno tych bezpośrednio wyrażonych, jak i wymagań *ukrytych*, czyli niekoniecznie związanych z bezpośrednim celem i przeznaczeniem aplikacji, a mimo to koniecznych dla zapewnienia prawidłowego funkcjonowania systemu. W tej części zostaną przedstawione sposoby oraz techniki pozyskiwania wymagań od klienta i innych udziałowców projektu.
- Planowanie i zarządzanie wymaganiami obejmuje planowanie procesu dokumentowania wymagań wraz z określaniem priorytetów dla implementacji poszczególnych wymagań oraz mechanizmami kontroli zmian.
- Dokumentowanie wymagań – opisuje, w jaki sposób specyfikować wymagania i na jakim poziomie szczegółowości, by mogły one zostać poprawnie zaimplementowane przez programistów.
- Komunikowanie wymagań, czyli udostępniania dokumentacji analitycznej udziałowcom projektu. Obejmuje również informowanie o zmianach w zakresie wymagań, treści wymagań etc.
- Ocena i walidacja rozwiązania polegająca na ocenie proponowanego rozwiązania oraz dostarczaniu wsparcia analitycznego implementacji oraz testowania.

Analiza przedsiębiorstwa

Analiza przedsiębiorstwa – pierwszy krok w poprawnej analizie biznesowej – jest elementem dość często pomijanym – jako że pozornie nie wnosi nic do procesu i nie daje wymiernych produktów. Tymczasem w wielu przypadkach poprawna identyfikacja wymagań klienta wymaga analizy procesów zachodzących w jego organizacji – czyli tak zwanego kontekstu biznesowego. Należy rozważyć:

- Czym zajmuje się organizacja?
- Komu i w jakim celu ma służyć system?
- Jaki jest cel wdrożenia projektowanego systemu? Co aplikacja ma robić, jakie procesy przedsiębiorstwa wspierać etc.
- Jakie procesy w przedsiębiorstwie ma obejmować system i z jakimi procesami będzie się komunikował?
- Czy system będzie komunikował się z innymi systemami wewnątrz i na zewnątrz organizacji klienta? Jeśli tak, to w jakim celu? Jakie informacje będą wymieniane?
- Elementy specyfiki branży – wymagania prawne i inne przepisy dotyczące obszaru, w którym działa organizacja klienta.

Informacje te pozwolą identyfikować wymagania dotyczące kontekstu systemu i umożliwią zaprojektowanie wstępnego modelu biznesowego na wysokim poziomie – jako defini-

cje celów i potrzeb organizacji klienta, i opisują, dlaczego został podjęty projekt i co ma osiągnąć oraz jakie metryki będą służyć do pomiaru postępu oraz powodzenia całości projektu.

W ramach analizy przedsiębiorstwa wykonywane jest również *planowanie strategiczne*. Podczas tego procesu rozważane są różne czynniki, okoliczności i potrzeby biznesowe – bieżące strategie związane z trendami rynkowymi, identyfikacja aktualnych kwestii biznesowych czy czynników przesądzających o konkurencyjności, dochodowości oraz efektywności organizacji.

Analiza charakterystyk strategicznych danego klienta umożliwia zrozumienie obszaru jego działalności, celów i dążeń – dzięki temu projektowanie nowego systemu będzie łatwiejsze. Analityk będzie też w stanie zaproponować taką koncepcję rozwiązania, która jak najlepiej dopasuje się do obecnej i planowanej strategii przedsiębiorstwa oraz istniejącej infrastruktury.

Planowanie i zarządzanie wymaganiami

Kolejny etap analizy – *planowanie i zarządzanie wymaganiami* – umożliwia określenie zasobów oraz zadań związanych z czynnościami planowania i zarządzania wymaganiami. Zadaniem analityka jest określenie, jakie czynności będą wykonywane w związku z gromadzeniem, analizą, zarządzaniem wymaganiami oraz w jaki sposób będą one realizowane. Analityk dostosowuje bieżące potrzeby projektowe do standardów swojej organizacji oraz przedsiębiorstwa klienta – proces planowania i zarządzania powinien być dopasowany do procesów, procedur i zwyczajów już obowiązujących w danej organizacji.

Do zadań analityka należy opracowanie *metodyki planowania i zarządzania wymaganiami*, która w możliwie największym stopniu gwarantuje, iż:

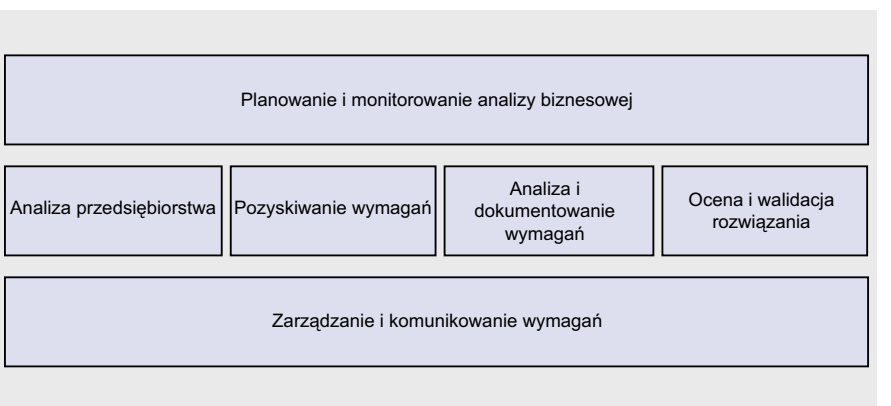
- Zidentyfikowano wszystkich udziałowców niezbędnych w procesie gromadzenia wymagań oraz zostały im przydzielone odpowiednie role, odpowiedzialności i zadania.
- Zadania związane z gromadzeniem wymagań są wykonywane efektywnie i zgodnie z planem oraz skoordynowane z innymi pracami wykonywanymi w ramach projektu.
- Zespół odpowiedzialny za planowanie i zarządzanie wymaganiami zna swoje role i rozumie zadania, za które jest odpowiedzialny. Każdy członek zespołu zna przydzielone sobie zadania i porządek ich realizacji.
- Analitycy są w stanie monitorować i reagować na wszelkiego typu zmiany w wymaganiach. Zmiany wdrażane są poprawnie i spójnie we wszystkich składnikach dokumentacji dotyczącej wymagań.

Identyfikacja *udziałowców* projektu jest kolejną ważną czynnością – wszystkie osoby zaangażowane w pozyskiwanie, tworzenie lub innego rodzaju przetwarzanie wymagań powinny zostać zidentyfikowane, a ich role oraz odpowiedzialności określone i udokumentowane. Podział ról odbywa się na podstawie planu projektu oraz innych dokumentów zarządczych dostępnych dla danego projektu. Znając udziałowców, analityk może lepiej zrozumieć, komu i w jakim celu ma służyć produkt – np. wiedząc, że użytkownikami końcowymi są pracownicy oddziałów bankowych (czyli *kasjerzy*), może przeanalizować czynności realizowane w środowisku docelowym – czyli placówce banku – i zaprojektować aplikację w taki sposób, by jak najlepiej wspierała ich pracę i codzienne zadania.

Planowanie i zarządzanie wymaganiami nie może odbywać się w oderwaniu od pozostałych czynności projektowych. Prace powinny zachodzić równolegle do prac projektowania wstępnej architektury oraz przygotowywania założeń do planu zapewnienia jakości. Analityk powinien udostępniać wyniki swoich prac na bieżąco innym członkom zespołu i zbierać od nich ewentualne uwagi i komentarze – dzięki temu często można znaleźć błąd w specyfikacji wymagań na wczesnym etapie i szybko wdrożyć poprawki.

Pozyskiwanie wymagań

Kolejny element analizy biznesowej – w wielu przypadkach najbardziej kluczowy i krytyczny dla powodzenia projektu – to proces pozyskiwania wymagań. Wymagania stanowią podstawę do opracowania i stworzenia rozwiązania IT zamówionego przez organizację klienta, dlatego też muszą być kompletne, jasne, jednoznaczne, poprawne oraz spójne. Co to oznacza w praktyce?



Rysunek 1. *Czynności analizy biznesowej*

Analiza przedsiębiorstwa

Analiza przedsiębiorstwa obejmuje grupę czynności podejmowanych na początku projektu polegających na określeniu danego biznesu w celu określenia kontekstu dla wymagań projektowych, projektu rozwiązania określonej inicjatywy oraz ustalenia planów długo- i krótkoterminowych. W większych projektach lub w przypadku organizacji klienta charakteryzujących się dużym stopniem złożoności procesów zachodzących wewnątrz niej, czynności związane z analizą przedsiębiorstwa stanowią oddzielny projekt.

Strategia to nie wszystko

Oprócz ustalania strategii, czyli bieżącego i przyszłego kursu organizacji, na etapie analizy przedsiębiorstwa wykonywane są również inne czynności. Obejmują one:

- tworzenie i utrzymywanie architektury biznesowej (wprowadzanie zmian, aktualizacji, uszczegółowień);
- opracowanie studium wykonalności w celu określenia optymalnego rozwiązania biznesowego;
- definiowanie zakresu projektu;
- przygotowanie przypadków biznesowych;
- wykonanie wstępnej analizy ryzyka;
- wsparcie podejmowania decyzji.

Osobą odpowiedzialną za realizację tych zadań ze strony dostawcy jest analityk biznesowy (przy czym niektóre zadania wykonywane są we współpracy z Project managerem, np. definiowanie zakresu projektu).

- Kompletność – zidentyfikowane wymagania pokrywają całość funkcjonalności projektowanej aplikacji. Proces biznesowy jest udokumentowany w szczegółach, wyjątki są określone, struktura danych opisana na wystarczająco niskim poziomie.
- Przejrzystość – sposób prezentacji treści i założeń wymagań jest prosty i zrozumiały dla odbiorców. Forma prezentacji umożliwia szybkie zapoznanie się z treścią wymagania i określenie obszaru / funkcjonalności, którego dane wymaganie dotyczy.
- Jednoznaczność – opis wymagania jednoznacznie określa sposób funkcjonowania, rozwiązanie problemu, technikę, którą należy zastosować etc. Wymóg jednoznaczności oznacza, iż treść musi być skonstruowana w taki sposób, by odbiorca interpretował ją zgodnie z zamierzeniem analityka – nie ma tu miejsca na domyślanie się, i rozpatrywanie alternatyw, które mogą wynikać z mało precyzyjnego tekstu wymagania.
- Poprawność – wymagania w żadnym wypadku nie powinny być błędne, czyli opisywać systemu niezgodnie z pożądanym stanem rzeczy. Błędy rzeczowe mogą wynikać zarówno z pośpiechu w dokumentacji wymagań, niedopatrzenia, czy zwykłej pomyłki, jak i braku wiedzy dziedzinowej czy nieznajomości przepisów i innych związanych rozporządzeń.
- Spójność – wymagania nie powinny się nawzajem wykluczać ani traktować podobnych problemów w odmienny sposób. Jeśli przykładowo wymaganie klienta mówi, iż podgląd wydruku powinien być generowany w postaci dokumentu PDF w formacie A4, analityk nie może opisywać funkcjonalności wydruku w innych wymaganiach inaczej – czyli np. jako generowanie edytowalnego dokumentu Word (chyba że rzeczywiście taka jest potrzeba klienta).

Planowanie i zarządzanie

Czynności planowania i zarządzania obejmują:

- określenie kluczowych ról;
- wybór czynności i zadań oraz ich organizację (określenie priorytetów, kolejności, powiązań i zależności);
- zarządzanie zakresem wymagań;
- komunikowanie bieżącego stanu procesu gromadzenia wymagań.

Udziałowcy projektu

- *Sponsor* – odpowiedzialny za projekt na poziomie zarządczym: pozyskiwanie funduszy, podejmowanie decyzji *go, no go,* wspierający zarządzanie zasobami poprzez zapewnienie odpowiedniej liczby zasobów.
- *Analityk biznesowy* – odpowiedzialny za pozyskiwanie, ewentualnie negocjowanie, analizę, dokumentowanie i przeglądy kontrolne wymagań oraz za prezentowanie wymagań pozostałym udziałowcom w celu uzyskania ich opinii i akceptacji.
- *Project manager* – zarządza projektem, dbając m.in. o to, by czynności związane z wymaganiami były wykonywane planowo, w przewidzianym budżecie oraz zgodnie z określonym zakresem. PM jest odpowiedzialny za upewnienie się, iż wymagania są zaakceptowane przez wszystkich uprawnionych udziałowców przed przejściem do kolejnych prac projektowych.
- *Deweloper* – rola obejmująca technicznych członków zespołu, zarówno programistów, jak i kierowników technicznych odpowiedzialnych za stworzenie projektu technicznego aplikacji. Programista implementuje wymagania (tworzy kod), w razie potrzeby zasięgając konsultacji analityka.
- *Specjalista QA* – odpowiedzialny za opracowanie, wdrożenie i nadzór standardów dotyczących zapewnienia jakości na różnych etapach projektu.
- *Architekt* – opracowuje projekt wysokiego poziomu oraz opisuje podejście do budowania architektury aplikacji. Architekt decyduje o tym, jaka architektura (projekt techniczny) najlepiej pasuje do danego projektu oraz spełni wymagania.
- *Analityk bazy danych* – odpowiedzialny za techniczne aspekty projektowania, tworzenia i zarządzania bazą danych projektu.
- *Analityk infrastruktury* – projektuje całą infrastrukturę i środowisko sprzętowe i softwarowe wymagane dla spełnienia wymagań dotyczących tworzenia oraz funkcjonowania systemu.
- *Analityk danych* – odpowiedzialny za ocenę wymagań dotyczących danych i informacji w danym systemie. Określa dane, które są wykorzystywane wielokrotnie, rozwiązuje problemy modelowania danych.
- *Właściciel (Solution owner)* – odpowiedzialny za zdefiniowanie i akceptację zakresu projektu i upewnienie się, iż pasuje on do strategii biznesowej. Akceptuje zmiany w zakresie projektu oraz określa kryteria powodzenia projektu i sposoby ich pomiaru.
- *Użytkownik końcowy* – ogólna nazwa grupy osób, które będą używać wyprodukowanej aplikacji. Użytkownik końcowy może być zarówno osobą z wewnątrz organizacji klienta (pracownik), jak i spoza organizacji (klient bezpośredni).
- *Ekspert dziedzinowy* – osoba dostarczająca ekspertyz w określonej dziedzinie biznesu. Ekspert wspiera definiowanie oraz akceptację wymagań funkcjonalnych projektu, zwykle współpracuje z analitykami biznesowymi w procesie identyfikacji i zarządzania wymaganiami. Eksperci często pochodzą z organizacji klienta lub są wynajmowani z firm konsultingowych – muszą to być osoby o szerokiej i dogłębnej wiedzy w danej dziedzinie. W mniejszych projektach, lub jeśli nie ma takiej konieczności (tj. wymagania i obszar dziedzinowy są jasne), ekspert dziedzinowy nie musi pojawiać się w zespole projektowym.

Analityk biznesowy powinien zapewnić takie środki i zastosować takie techniki, by zapewnić jak największą efektywność procesu pozyskiwania wymagań oraz wysoką jakość uzyskanych produktów. Pozyskiwanie wymagań nie jest zamkniętym, skończonym etapem odizolowanym od pozostałych faz projektu – wymagania określone na etapie początkowym (podczas analizy) mogą okazać się niekompletne na etapie projektowania szczegółowego (dokumentowania specyfikacji funkcjonalnej) lub implementacji.

W procesie pozyskiwania wymagań należy rozpocząć od zidentyfikowania wymagań głównych, a następnie je stopniowo uszczegóławiać. Podejście takie umożliwia stosunkowo szybkie opracowanie modelu procesów biznesowych w aplikacji, który daje podgląd na ogólną budowę i funkcjonalność projektowanego systemu. Istnieje również podejście odwrotne – od szczegółu do ogółu – polegające na zidentyfikowaniu szczegółowych, elementarnych wymagań w pierwszej kolejności, a następnie grupowanie ich w wymagania na wysokim poziomie. Ta metoda nie sprawdzi się jednak w przypadku, gdy klient ma wizję ogólnej funkcjonalności systemu, bez szczegółowych wymagań dotyczących poszczególnych elementów (takich jak ekrany aplikacji, reguły związane z walidacją danych, dokładne algorytmy obliczeniowe).

Skąd natomiast brać wymagania? Klient nie zawsze dostarcza kompletny katalog wymagań, na podstawie którego można łatwo zidentyfikować główne wymagania i przejść do ich uszczegóławiania i precyzowania. Pozyskiwanie wymagań odbywa się na podstawie:

- Założeń oferty handlowej – główne funkcje systemu powinny zostać wyspecyfikowane w ofercie handlowej lub zapisach umowy. Funkcje te są poddawane analizie, uszczegóławiane i rozbijane na poszczególne funkcjonalności czy przypadki użycia.
- Warsztatów analitycznych, które przeprowadzane są przy udziale przedstawicieli biznesu klienta. W trakcie warsztatów dostawca przedstawia propozycje rozwiązań – dotyczących bądź to określonych funkcjonalności czy ich aspektów, bądź systemu jako całości – klient natomiast je weryfikuje i / lub uszczegóławia.

- Informacji udzielonych przez uprawnionych do tego pracowników klienta w *formie pisemnej* lub potwierdzonych drogą pisemną.
- Analizy istniejących systemów klienta i identyfikacji działających już w nich mechanizmów i rozwiązań. Metoda ta jest szczególnie korzystna i wydajna w przypadku, gdy klient pragnie, by wszystkie jego systemy działały na podobnej zasadzie (co oczywiście nie oznacza, iż mają być takie same) – np. stosowały te same mechanizmy zapisywania, przeglądania informacji czy prezentacji danych. Zdarzają się sytuacje, w których klient wręcz wymaga, by nowy system *pasował wizualnie* i funkcjonalnie do już istniejących. W tej sytuacji najlepszą metodą jest analiza dotychczasowych rozwiązań, identyfikacja wspólnych mechanizmów, które da się przenieść do projektowanego systemu i udokumentowanie ich w specyfikacji.

W pozyskiwaniu wymagań (i oddzielaniu wymagań od zbędnych informacji, które nie wnoszą żadnej wartości do procesu analizy), analityk stosuje różne techniki – wywiady, spotkania wyjaśniające, warsztaty etc (Tabela 1). Jednak oprócz techniki analityk powinien posiadać pewne umiejętności – jako że nie każdy ma predyspozycje do pracy w takiej roli. Umiejętności te to między innymi: zdolność zdobywania i oceny informacji pod kątem ich przydatności w określonym aspekcie oraz obserwacji zdarzeń i faktów i wyciągania wniosków z informacji, myślenie abstrakcyjne i twórcze, znajdowanie alternatyw dla rozwiązania danego problemu. Prócz wymienionych cech analityk musi być osobą, która potrafi rozwiązywać konflikty i negocjować oraz posiadać doskonałe umiejętności słuchania i komunikacji werbalnej. Człowiek posiadający szeroką wiedzę na temat analizy biznesowej i modelowania raczej nie nadaje się na analityka, jeżeli nie potrafi dyskutować, słuchać, co mają mu do przekazania inni ludzie lub ma problemy z jasnym wyrażeniem swoich myśli.

Tabela 1 prezentuje kilka najczęściej stosowanych technik pozyskiwania wymagań – wybór odpowiedniej jest uzależniony od potrzeb i możliwości danego projektu, jak również od doświadczenia i umiejętności analityka.

Analiza i dokumentowanie wymagań

Wymagania pozyskane należy poddać analizie i udokumentować w odpowiedniej, ustalonej wcześniej formie. *Analiza i dokumentowanie wymagań* polega na określeniu zależności pomiędzy wymaganiami, zaprojektowaniu struktury funkcjonalnej systemu oraz opisaniu wymagań w formie ustrukturyzowanej, określonej zasadami tworzenia specyfikacji i umożliwiającej efektywne wykorzystanie w procesie projektowania technicznego i implementacji systemu. Celem dokumentacji wymagań jest określenie i opisanie właściwości biznesowych rozwiązania w taki sposób, by wszyscy członkowie zespołu wiedzieli, jak je zaprojektować i zaimplementować.

Jak już wspominano wcześniej, wymagania dostarczone w formie listy czy katalogu wymagań nie są wystarczającą podstawą do sfinalizowania procesu analizy, a dalej do implementacji; nie mogą być również jedynym źródłem dokumentacji testowej. Wymagania należy opisać na odpowiednim poziomie szczegółowości tak, by dokumentacja analityczna dawała obraz całej funkcjonalności systemu oraz przekazywała idee procesów biznesowych, jakie będą zachodzić w danej aplikacji. Jednym z podejść do dokumentowania systemu jest podejście zstępujące, czyli rozpoczęcie prac z modelem biznesowym na wysokim poziomie (np. diagram BPMN), który w następnej kolejności jest rozbijany na poszczególne składniki i stopniowo uszczegóławiany..

Tabela 1. *Techniki pozyskiwania wymagań*

Technika	Opis
Burza mózgów	Technika polegająca na wspólnym tworzeniu i analizowaniu pomysłów podczas grupowego spotkania. Uczestnicy dostarczają pomysłów na rozwiązanie postawionego problemu, pomysły owe spisuje się, analizuje, po czym wybiera rozwiązanie akceptowane przez ogół.
Analiza dokumentacji	Polega na przeglądaniu istniejącej w organizacji dokumentacji celem identyfikacji wymagań obecnych i potencjalnych. Technika stosowana w celu określenia istniejących zasad biznesowych, elementów i atrybutów, które należy zawrzeć w nowym systemie lub które będą musiały być zmienione z powodu wdrażania nowego systemu. Metoda może być również wykorzystywana w przypadku, gdy eksperci danej dziedziny biznesowej nie są dostępni dla konsultacji analitycznych i analityk musi samodzielnie poznawać określone zagadnienie.
Analiza interfejsów	Analiza zewnętrznych interfejsów – zarówno interfejsu użytkownika, jak i interfejsów do / z zewnętrznych systemów lub urządzeń. Analiza taka pomaga określić granice systemu oraz stwierdzić, jakie dane i jakie funkcje przepływają pomiędzy systemami. Analiza interfejsów powinna dać w wyniku dokument słownika danych, który gromadzi wszystkie dane wymieniane pomiędzy systemami w sposób spójny i umożliwiający ich kontrolowanie oraz utrzymywanie w aktualnym stanie.
Wywiad	Technika polegająca na zadawaniu pytań dotyczących określonego obszaru działania danej osoby, procesu, problemu. Wywiad przeprowadzany jest przez analityka, zaś osobą pytaną jest ekspert dziedzinowy lub użytkownik końcowy projektowanego rozwiązania. Wywiad może być zarówno ustrukturyzowany – kiedy analityk posiada przygotowany wcześniej zestaw pytań; lub nie ustrukturyzowany – polegający na dyskusji dotyczącej oczekiwań wobec projektowanego rozwiązania. Ustalenia i wnioski z wywiadu powinny być udokumentowane.
Obserwacja	Technika ta polega na wyciąganiu wniosków na podstawie obserwacji pracy ludzi wykonujących swoje zadania. Umożliwia dokładne poznanie przebiegu określonego procesu, procedur, czynności, produktów oraz nieformalnych zwyczajów, zwykle nie umieszczanych w dokumentacji przedsiębiorstwa a wpływających na przebieg pracy. Obserwacja może być pasywna - kiedy analityk jedynie obserwuje osobę wykonującą swoja pracę, bez zadawania dodatkowych pytań; lub aktywna – polegająca na obserwacji połączonej z dyskusją z pracownikiem i omawianiem bieżących czynności.
Prototypowanie	Prototypowanie – zastosowane jako technika pozyskiwania wymagań – umożliwia wykrycie wymagań dotyczących UI systemu jeszcze przed oficjalnym rozpoczęciem prac implementacyjnych. Prototypy aplikacji (tworzone np. jako makieta HTML) pozwalają na wizualizację wstępnego projektu analityka i weryfikację rozwiązania przez klienta. Klient może sprawdzić, czy bieżący projekt spełnia jego oczekiwania, dodać nowe pomysły i rozwiązania itp. Zwykle w trakcie pracy z prototypem zostają odkryte nowe wymagania (np. dotyczące użyteczności, ergonomii), które zostają uszczegółowione bądź w kolejnej wersji prototypu (prototypowanie ewolucyjne), bądź włączone bezpośrednio do specyfikacji analitycznej.
Warsztaty	Warsztaty wymagań stanowią ustrukturyzowaną metodę pozyskiwania wymagań. warsztaty służą do określania zakresu, identyfikacji, szczegółowego definiowania i ustalania priorytetów wymagań projektowanego systemu. Dobrze zorganizowany warsztat to jedna z najlepszych technik określania wymagań – ponieważ oznacza dobrą komunikację i porozumienie pomiędzy udziałowcami projektu (decyzje podjęte podczas warsztatów są dyskutowane i akceptowane przez wszystkich członków).

Zwykle proces analizy i dokumentacji dzieli się na kolejne iteracje, w których produkty jednej iteracji stanowią wejście do iteracji kolejnej i są uszczegóławiane i kompletowane na kolejnych etapach. Po zakończeniu iteracji, dotychczasowe rezultaty są weryfikowane wewnętrznie (przez zespół analityków, QA oraz programistów) oraz zewnętrznie (po pozytywnej kontroli wewnętrznej) przez przedstawicieli klienta.

W trakcie analizy i dokumentacji wymagań analityk biznesowy rozwija dotychczas określone wymagania, dodaje nowe (zgodnie z zakresem projektu) i opracowuje spójny, kompletny model analityczny systemu.

Ukończenie każdego etapu prac powinno kończyć się walidacją produktów ze strony udziałowców projektu i akceptacją rezultatów bądź przekazaniem wyników do poprawy / uzupełnienia. Szczególnie istotną kwestią jest prawidłowa komunikacja w procesie oraz zapewnienie, iż wszyscy członkowie zespołu projektowego (zarówno ze strony dostawcy, jak i klienta) znają wyniki prac oraz są świadomi, czego od nich oczekuje analityk. A analityk oczekuje:

- Od *programistów* – weryfikacji możliwości implementacji rozwiązania, analizy specyfikacji funkcjonalnej pod kątem ewentualnych braków szczegółów uniemożliwiających poprawną implementację oraz możliwych problemów.
- Od *zespołu QA* – weryfikacji produktów analizy pod kątem błędów logicznych, edytorskich, kontroli spójności, kompletności i poprawności.
- Od *przedstawicieli biznesu klienta* – weryfikacji produktów analizy pod kątem zgodności z oczekiwaniami i stopnia spełnienia określonych wymagań.

Ilość iteracji zależy od rozmiaru aplikacji (liczby funkcjonalności, modułów itp.) i powinna gwarantować uzyskanie rezultatów na odpowiednim poziomie jakości – produkty każdej iteracji powinny być poddane przeglądowi i skorygowane przed dostarczeniem wersji finalnych. Nie zaleca się zbyt dużej liczby iteracji – powoduje to wstrzymanie postępu projektu i częściowo blokuje np. proces implementacji. Z kolei niewystarczająca liczba iteracji może oznaczać brak czasu i możliwości szczegółowego opisania wszystkich elementów systemu, wykonania wewnętrznej i zewnętrznej weryfikacji oraz poprawy wszystkich znalezionych i zakomunikowanych usterek.

Do analizy wymagań można zastosować kilka technik – graficzne, tekstowe czy oparte na macierzach. Analityk powinien potrafić posługiwać się więcej niż jedną – preferowaną przez siebie – techniką, co umożliwia efektywniejsze i pełniejsze modelowanie i dokumentowanie wymagań, a także rozumienie zawartości produktów wytworzonych przez innych analityków (w ramach zespołu projektowego i poza nim – np. czytanie dokumentacji związanej z danym biznesem, lecz nie odnoszącej się bezpośrednio do projektu).

Wybór odpowiedniej techniki leży zwykle w gestii analityka (choć zdarzają się sytuacje, kiedy klient narzuca stosowanie określo-

Sesja warsztatowa – o czym należy pamiętać?

- Ustal termin warsztatów odpowiednio wcześniej (z wyprzedzeniem co najmniej tygodniowym), by uczestnicy mogli przygotować się do sesji.
- Upewnij się, że osoby, których obecność jest niezbędna podczas warsztatu, są dostępne w określonym terminie. W przeciwnym razie przesuń termin sesji lub zaproś osoby, których kompetencje odpowiadają nieobecnym w sesji.
- Zaproszenie na warsztaty powinno obejmować temat i ramowy przebieg sesji. W agendzie warsztatu określ cel, tematy, jakie będą omówione na spotkaniu oraz planowany czas spotkania i – w miarę możliwości - czasy przeznaczone na poszczególne składniki sesji.
- Przekaż uczestnikom sesji listę problemów i zagadnień do omówienia *przed* ustalonym terminem warsztatów. Uczestnicy, a w szczególności przedstawiciele klienta, powinni mieć czas zapoznać się z owymi kwestiami, by podczas sesji przedstawić własne rozwiązanie lub wyjaśnienie do pytań.
- Określ rolę prowadzącego warsztaty – osobę odpowiedzialną za notowanie ustaleń z warsztatów (zarówno ustaleń dotyczących wymagań, jak i innych kwestii). Zwykle jest to analityk biznesowy.
- Dokumentuj lub upewnij się, że taka odpowiedzialność została ustalona, wszystkie ustalenia i decyzje podjęte podczas warsztatów. Dla każdego zagadnienia wymagającego podjęcia odpowiednich akcji muszą być wyznaczone osoby odpowiedzialne za ich realizację. Notatki ze spotkania przekaż wszystkim uczestnikom sesji oraz osobom, które powinny być poinformowane o ustaleniach i podjętych decyzjach.
- Wspólnie z pozostałymi uczestnikami sesji określ terminy, w których należy wykonać ustalone zadania, ponadto ustal sposoby odbioru rezultatów zadań.

Analiza wymagań – problemy

Jednym z problemów pojawiających się w analizie i dokumentacji wymagań są luki w wymaganiach dostarczonych przez udziałowców. Zadaniem analityka – oraz wspierających go członków zespołu – jest zidentyfikowanie jak największej liczby luk, niezgodności i wątpliwości, by w następnej kolejności ustalić ich rozwiązanie z innymi udziałowcami projektu i udokumentować kompletne i spójne wymagania. Analityk powinien prowadzić listę braków, na której zapisuje wszelkie uwagi do obecnie dostępnych wymagań i w odpowiednim terminie omawiać treść listy z przedstawicielem biznesu klienta celem znalezienia rozwiązania spełniającego wymogi i oczekiwania klienta.

Komunikowanie wymagań – problemy

Doskonałe produkty i informacje wejściowe nie zawsze są gwarancją sukcesu projektu informatycznego. Tam, gdzie zawodzi komunikacja, zwykle nie da się zrealizować przedsięwzięcia zgodnie z planem. W przypadku analizy biznesowej częstymi błędami w komunikacji są:

- Brak informacji o nowych wersjach i zmianach w dokumentacji – co powoduje np., że programiści implementują kod zgodnie z nieaktualną specyfikacją, testerzy weryfikują system wedle nieodpowiedniej dokumentacji i pojawia się ogólna konsternacja.
- Małe zaangażowanie analityków we wsparcie implementacji i testowania – jakże często zdarza się, że programista czy tester nie rozumiejąc do końca danej funkcjonalności, prosi analityka o wyjaśnienie i odpowiedź na swojego maila otrzymuje po paru dniach czy tygodniach... kiedy dawno minął już termin wykonania jego zadania.
- Informacje udzielane ustnie – to poważny problem, występujący, gdy dokumentacja analityczna nie jest wystarczająco szczegółowa, by wyjaśniać wszystkie aspekty funkcjonalności. Zdarza się, że dokumentacja podaje tylko podstawowy opis jakiejś funkcji, bez szczegółów biznesowych etc. i aby poprawnie zaimplementować ową funkcję, programista musi dopytywać o szczegóły. Jeśli wyjaśnienie nadchodzi drogą pisemną, jest znane innym zainteresowanym (np. testerom) i pozostawia *dowód rzeczowy* w postaci maila – pół biedy. Gorzej, jeśli analityk wyjaśnia daną kwestię ustnie jedynie pytającemu – nie ma żadnej gwarancji, że pytający zrozumiał dokładnie, jak ma działać dana funkcja (a w przypadku błędnej implementacji nie udowodni, że stworzył kod zgodnie z wolą analityka), ponadto wyjaśnienie nie jest znane pozostałym członkom zespołu, na których pracę taka informacja może mieć wpływ.
- Nieefektywna współpraca pomiędzy analitykami – jest też formą komunikacji: w przypadku dużych zespołów analitycznych sprawna komunikacja i współdziałania to czynnik krytyczny. Analitycy muszą informować się na bieżąco o statusie wykonanych prac, problemach, które wystąpiły, zmianach w wymaganiach, o których są poinformowani, a które mogą mieć wpływ na przebieg lub zakres pracy innych analityków.

nej techniki) i zależy od organizacji i standardów firmy dostawcy, skali projektu oraz subiektywnej (lecz popartej doświadczeniem) opinii analityka.

Komunikowanie wymagań

Prawidłowy proces analizy biznesowej wymaga sprawnej komunikacji. *Komunikowanie wymagań* to bardzo istotny element – bowiem nawet najdoskonalsza dokumentacja analityczna jest bez znaczenia, jeśli członkowie zespołu nie wiedzą, jak do niej dotrzeć, nie znają aktualnych wersji i zmian. Komunikowanie wymagań odbywa się zwykle za pomocą repozytorium plików, najlepiej z możliwością kontroli wersji (np. VSS).

Wytworzone dokumenty należy umieszczać w określonym miejscu bazy danych, do którego dostęp mają wszyscy członkowie zespołu projektowego, przy czym poszczególne składniki dokumentacji powinny znajdować się w odpowiednich katalogach jednoznacznie określających np. typ i status dokumentów, obszar funkcjonalny etc.

Gromadzenie i zarządzanie treścią wymagań odbywa się zwykle za pomocą listy (katalogu) wymagań umieszczonej oraz za pomocą katalogu zawierającego poszczególne dokumenty wymagań (o wyższym stopniu szczegółowości, niż w przypadku listy). W przypadku listy wymagań, każde wymaganie powinno mieć zdefiniowany:

- numer referencyjny (służący np. do wykazania listy spełnionych wymagań w określonej, szczegółowej specyfikacji funkcjonalnej);
- treść;
- priorytet (`MUST`, `SHOULD`, `COULD`), który umożliwia planowanie kolejności dokumentowania, a dalej implementacji wymagań;
- opis spełnienia (w jaki sposób dostawca ma zamiar spełnić dane wymaganie).

Miejsce przechowywania i zarządzania dokumentacją analityczną powinno być określane na etapie inicjacji projektu, opisane w planie komunikacji i znane wszystkim członkom zespołu projektowego. Jak najwcześniej należy upewnić się, że personel projektowy ma dostęp do repozytorium i dokumentów, przy czym jedynie analitycy (twórcy dokumentacji) mają uprawnienia do modyfikacji treści dokumentów. Pozostali pracownicy powinni mieć prawa tylko do odczytu – a to dlatego, by uniknąć przypadkowych modyfikacji treści dokumentów przez osoby do tego nieuprawnione.

Każda istotna zmiana w wymaganiach czy dokumentacji analitycznej powinna być zakomunikowana członkom zespołu – najlepiej za pomocą poczty elektronicznej. Autorem informacji o zmianie powinien być analityk, który dokonał zmiany lub – w większych zespołach analitycznych – kierownik zespołu. W treści powiadomienia należy umieścić krótki opis zmiany, listę wymagań lub funkcjonalności, których zmiana dotyczy oraz miejsce przechowywania i nazwę aktualnej dokumentacji. Dobrą praktyką jest też umieszczanie odnośnika do miejsca przechowywania określonego dokumentu tak, by odbiorcy wiadomości mogli od razu zapoznać się ze zmianami. Przykładowo: system Lotus Notes umożliwia nie tylko tworzenie całej struktury dokumentacji projektowej, ale i wysyłanie automatycznych powiadomień do wszystkich członków zespołu w przypadku modyfikacji treści dokumentu. Takie powiadomienie zawiera również link wiodący bezpośrednio do danego dokumentu.

Walidacja rozwiązania

Analityk biznesowy jest – ze strony producenta oprogramowania – najlepiej zorientowanym w funkcjonalności projektowanego systemu członkiem zespołu. Jest zatem najbardziej odpowiednią osobą do udzielania wyjaśnień odnośnie funkcji i możliwości aplikacji oraz weryfikacji produktów poszczególnych etapów.

Ponadto wsparcie analityczne obejmuje podział prac projektowych wchodzących w skład danej fazy na mniejsze elementy – łatwiejsze do udokumentowania oraz późniejszej implementacji, a także kontrolę produktów analizy technicznej pod względem zgodności z projektem funkcjonalnym.

Jeśli projektowane oprogramowanie wykorzystuje gotowe komponenty lub zawiera gotowe moduły pochodzące z produktu, analityk bierze udział w pracach związanych z dostosowaniem istniejącego systemu do wymagań klienta oraz z identyfikacją i projektowaniem wymagań dotyczących interfejsów.

Podczas prac programistycznych analityk powinien czynnie udzielać wsparcia mery-

Walidacja rozwiązania – problemy

Niekiedy zdarza się, że prace analityczne są niemal całkowicie oddzielone od pozostałych czynności projektowych – analitycy modelują wymagania, przygotowują dokumentację użytkowników i na tym się kończy ich odpowiedzialność. Może to wynikać zarówno z braku czasu i możliwości, jeśli wsparcie analityczne nie zostało ujęte w harmonogramie prac implementacyjnych i testowych, jak i w nieświadomości, jak duże znaczenie ma systematyczne wsparcie analityczne dla prawidłowego przebiegu projektu. Tymczasem brak wsparcia ze strony BA może powodować następujące problemy:

- błędna interpretacja wymagań przez programistów i stworzenie kodu, który nie będzie spełniał oczekiwań klienta;
- przypadki testowe niepokrywające krytycznych funkcji aplikacji w odpowiednim stopniu;
- straty czasu wynikające z niewłaściwego wykonywania testów i raportowania ich wyników (m.in. konieczność analizy nieprawidłowych zgłoszeń błędów);
- niewłaściwe podejście do testowania przyjęte przez zespół klienta – wynikające z niezrozumienia funkcjonalności i zakresu aplikacji oraz traktowania zmian jako błędów.

Pojęcia

- *Architektura biznesowa* – koncepcja rozwiązania systemu implementująca wszystkie założenia i aspekty biznesowe – np. przepływ procesów, interakcje z innymi systemami, przepływ informacji biznesowej.
- *BA – Business Analysis* – analiza biznesowa; *Business Analyst* – analityk biznesowy.
- *Modelowanie* – reprezentacja biznesu lub danego systemu. Zwykle prezentowana w postaci graficznej uzupełnionej tekstem.
- *Ograniczenie* – każde ograniczenie, biznesowe czy techniczne nałożone na system (np. limit budżetowy dla projektu, ograniczenia związane z określoną architekturą systemu).
- *Produkt* – rozwiązanie problemu biznesowego, system będący wynikiem projektu informatycznego.
- *Projekt* – czasowe przedsięwzięcie mające na celu stworzenie unikalnego produktu, usługi lub innego rezultatu. Ograniczone jest budżetem, dostępnością zasobów (np. ludzkich, sprzętowych) oraz kosztami.
- *Projekt funkcjonalny* - zachowanie produktu (systemu) możliwe do obserwacji.
- *QA – Quality Assurance* – zapewnienie jakości; termin odnoszący się do ogólnego procesu planowania, realizacji odpowiednich działań zapobiegawczych i korygujących oraz kontroli jakości zarówno w danym projekcie, jak i przedsiębiorstwie jako całości.
- *Wymaganie* – warunek lub właściwość systemu wymagane przez udziałowców dla rozwiązania danego problemu lub uzyskania celu.
- *Wymaganie funkcjonalne* – wymaganie, które opisuje zachowanie systemu oraz informację przetwarzaną przez system.
- *Wymaganie niefunkcjonalne*- wymaganie, które opisuje możliwości systemu niezwiązane z funkcjonalnością – np. bezpieczeństwo, liczba użytkowników, którzy mogą równocześnie pracować z systemem, użyteczność etc.

torycznego - co polega m.in. na udzielaniu wyczerpujących wyjaśnień wszędzie tam, gdzie opis dokumentacyjny nie wystarcza, by w pełni zrozumieć wymagania i funkcjonowanie systemu, lub gdy programista nie ma dużego doświadczenia w projektach określonego typu. Plan projektu powinien uwzględniać odpowiedni bufor czasowy na tego typu wsparcie – programista na każdym etapie swoich prac powinien mieć możliwość zasięgnięcia opinii analityków odnośnie poprawności i kompletności implementacji. Taka praktyka może znacznie usprawnić prace implementacyjne oraz podnieść jakość produktu – należy bowiem mieć na względzie, że w dużych projektach informatycznych oraz wszędzie tam, gdzie projektowany system dotyczy specyficznej branży (wymagającej głębokiej wiedzy specjalistycznej), dokumentacja analityczna może nie być wystarczająco szczegółowa dla poprawnej implementacji. Analityk może dokumentować wymagania bez opisywania wszystkich detali i reguł biznesowych, może też odwoływać się do dokumentacji zawierającej specyficzne informacje. Programista, nieorientujący się w całości dokumentacji, nie zawsze wie, gdzie może znaleźć informacje, których potrzebuje. W takich sytuacjach analityk służy wyjaśnieniami i precyzuje, w jaki sposób zaimplementować daną funkcjonalność. Dodatkową weryfikację należy przeprowadzić po ukończeniu kodowania określonej funkcji czy modułu systemu – wtedy zarówno analityk, jak i programista będą mieli niemal całkowitą pewność, że wymagania klienta zostały prawidłowo spełnione.

Plan projektu nie zawsze pozwala na stałe konsultacje analityczne i systematyczne wsparcie procesu kodowania – zwykle analitycy są obciążeni pracą w stopniu uniemożliwiającym poświęcenie czasu na takie czynności. Warto jednak przynajmniej zastosować praktykę przeglądania produktów implementacji *po zakończeniu każdego etapu prac* – dzięki takiemu podejściu, znacząco maleje ryzyko błędów wynikających z błędnej implementacji wymagań i kumulowania się defektów w fazie testowania.

Ocena rozwiązania nie ogranicza się do weryfikacji produktu w trakcie implementacji. Po stworzeniu stabilnej wersji systemu gotowej do przeprowadzenia testów, analityk biznesowy wspiera prace zespołu QA. Analityk może wspierać planowanie i przygotowanie procesu testowego – określając wymagania, jakie muszą być pokryte przypadkami testowymi, czy też weryfikując produkty testów – plany, przypadki i scenariusze testowe. Celem tej czynności jest upewnienie się, iż projekt testów zapewnia maksymalne pokrycie funkcjonalności przypadkami testowymi i minimalizację ryzyka biznesowego związanego z dostarczeniem nieprawidłowo działającego produktu. W trakcie wykonywania testów analitycy udzielają wyjaśnień w kwestiach niejasnych oraz są odpowiedzialni za rozwiązywanie problemów związanych z funkcjonalnością aplikacji – np. wynikających z braków w dokumentacji analitycznej.

Niekiedy analityk biznesowy może być na stałe przydzielony do prac testowych jako osoba odpowiedzialna za weryfikację zgłoszeń błędów napływających od testerów – zwłaszcza w przypadku, gdy testy wykonują pracownicy klienta, niezbyt dobrze znający funkcjonalność aplikacji. W takiej sytuacji zadaniem analityka będzie podejmowanie nowych zgłoszeń, analiza zgłoszenia i weryfikacja, czy opisuje ono błąd, żądanie zmiany lub nie jest zasadne w ogóle oraz przekazanie zgłoszenia do naprawy – lub odrzucenie w uzasadnionych wypadkach.

W późniejszej fazie projektu analitycy często wspierają planowanie, przygotowanie i wykonanie testów akceptacyjnych (UAT) oraz raportowanie błędów. Niemal normą jest, iż analityk biznesowy (zwykle Główny analityk) bierze aktywny udział (lub wręcz samodzielnie) w planowaniu testów UAT, określaniu ich zakresu, kryteriów wejścia i wyjścia oraz przygotowaniu zestawu scenariuszy. Analityk – jako osoba projektująca dany system – ze strony dostawcy wie najwięcej o funkcjonalności rozwiązania oraz możliwych sposobach weryfikacji jego działania, dlatego też ma największe kompetencje, by organizować lub wspierać organizację testów użytkowników.

Podsumowanie

Według badania *Standish Group*, trzy najważniejsze powody; dla których upadają projekty informatyczne, to:

- Brak wkładu użytkowników: 12,8%;
- Niekompletne wymagania lub specyfikacja: 12,3%;
- Zmieniające się wymagania lub specyfikacja: 11,8%.

Dane te jednoznacznie wskazują, że czynniki związane z produktami analizy biznesowej stanowią główne wyznaczniki ryzyka projektowego i mogą przesądzić o sukcesie lub upadku przedsięwzięcia. Celem projektu jest dostarczenie oprogramowania zgodnego z oczekiwaniami klienta – niemożliwe jest więc wykonanie zlecenia bez dogłębnej znajomości owych oczekiwań i poprawnego przełożenia ich na możliwości systemu informatycznego. Zadaniem tym zajmuje się właśnie analiza biznesowa. Dobrze zorganizowany i zrealizowany proces analizy umożliwia pozyskanie szczegółowych wymagań co do sposobu funkcjonowania i możliwości projektowanej aplikacji – zarówno wymagań uświadomionych (czyli takich, z których istnienia klient zdaje sobie sprawę), jak i pośrednich – pozornie niezwiązanych z głównym celem i przeznaczeniem systemu, a mimo to wpływających na jego prawidłowe i bezpieczne działanie.

Niniejszy artykuł przedstawił zasady organizacji procesu analizy w projekcie informatycznym oraz czynności wchodzące w skład poszczególnych etapów. Czytelnik dowiedział się, co należy wykonać w ramach pozyskiwania i modelowania wymagań systemowych, jak ulepszyć komunikację w projekcie oraz w jaki sposób zwiększyć szanse na sukces przedsięwzięcia za pomocą zaangażowania członków zespołu BA w prace programistyczne i testowe. Nie wszędzie można zastosować wszystkie opisane w niniejszym tekście techniki analizy, nie zawsze da się zaplanować proces analizy tak, by objął wszystkie przedstawione czynności – w gestii czytelnika leży dobór odpowiednich mechanizmów i organizacja procesu analizy w sposób najbardziej pasujący do danego typu przedsiębiorstwa oraz specyfiki projektu.

W Sieci

- *http://www.erudis.pl/pl/artykuly* – artykuły związane z modelowaniem w notacji UML;
- *http://www.aion.com.pl/analiza_biznesowa.html* – artykuły dotyczące modelowania wymagań i dokumentowania procesów;
- *http://www.uml.com.pl/modules/articles/article.php?id=18* – artykuł o przejściu od analizy biznesowej do analizy systemowej;
- *www.theiiba.org* – witryna M iędzynarodowego Instytutu Analizy Biznesowej. Wśród wielu informacji dotyczących analizy biznesowej udostępnia również BABOK – czyli Business Analysis Body of Knowledge stanowiący analityczny odpowiednik PMBOK.

KAROLINA ZMITROWICZ

Pracuje na stanowisku Analityka biznesowego w firmie Profi-Data. Karolina specjalizuje się obecnie w modelowaniu wymagań biznesowych. Wcześniej pracowała jako Manager Quality Assurance w projektach informatycznych w sektorze finansowo – bankowym.

Profi-Data to producent oprogramowania specjalizujący się w budowie dedykowanych rozwiązań informatycznych dla ubezpieczeń, bankowości i administracji publicznej. Firma posiada certyfikat jakości ISO 9001:2000 w zakresie produkcji oprogramowania.

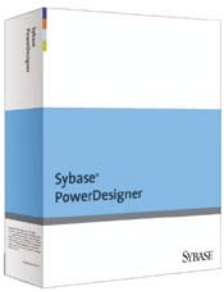
Kontakt z autorem: karolina_zmitrowicz@wp.pl

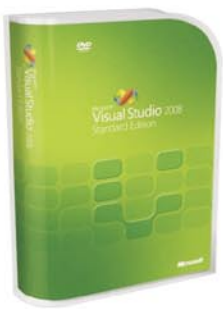
TTS Company

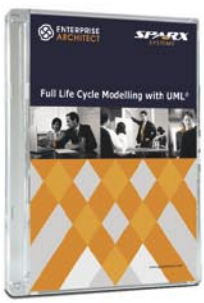
Qt Reseller

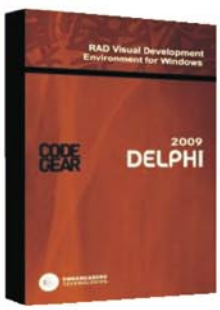
Największy wybór profesjonalnego oprogramowania w Polsce !

... w ofercie produkty ponad 200 producentów ...









www.OprogramowanieKomputerowe.pl

Code less. Create more. Deploy everywhere.

Więcej informacji: (022) 868 40 42

Sprzedaż ▸▸ Dystrybucja ▸▸ Import na zamówienie

TTS Company Sp. z o.o. ul.Słowicza 53 02-170 Warszawa e-mail: tts@tts.com.pl www.tts.com.pl

Monitorowanie oprogramowania

Lepiej zapobiegać, niż leczyć...

Coraz częściej instytucje zamawiające usługi informatyczne posiadają własne departamenty IT i pragną w określonym zakresie uczestniczyć w procesie wytwarzania oprogramowania. Niniejszy artykuł przedstawia najczęściej spotykane z punktu widzenia klienta problemy związane z taką współpracą oraz propozycje rozwiązania czy uniknięcia owych problemów.

Dowiesz się:

- Jak skonstruować kontrakt z dostawcą, by zagwarantować pożądaną jakość procesu wytwarzania oprogramowania.

Powinieneś wiedzieć:

- Podstawowa wiedza z zakresu testowania i procedur odbioru oprogramowania;
- Podstawowa wiedza z zakresu zarządzania projektem informatycznym.

Poziom trudności

Do lamusa odchodzą czasy, w których zamawiający oprogramowanie nie miał pojęcia o jakości procesu i produktu informatycznego i ślepo wierzył w zapewnienia deklarowane przez dostawcę IT. Obecnie większość instytucjonalnych klientów posiada własne działy IT, wykwalifikowany personel i świadomość procesów towarzyszących wytwarzaniu oprogramowania. W wielu przypadkach klient wie dokładnie, czego może oczekiwać od dostawcy, i w jaki sposób weryfikować produkty projektu. Pracownicy klienta mają wiedzę i doświadczenie z zakresu realizacji projektów IT, technik analizy i testowania – zdarza się, iż równie duże (a nawet większe), niż personel dostawcy oprogramowania. Jednak nie zawsze klient wie, w jaki sposób zapewnić sobie środki i możliwości wglądu w proces produkcji systemu, czy jak egzekwować swoje prawa.

W następnej części artykułu czytelnik zapozna się z najczęściej spotykanymi problemami we współpracy klienta z dostawcą IT.

Wgląd w testy wewnętrzne

Klienci często uskarżają się na to, że dostawcy nie dają im wglądu w swoje testy – co klient odbiera jako próby ukrycia rzeczywistych wyników testów oraz postępu prac (na przykład stopnia pokrycia funkcjonalności testami). Dostawca tłumaczy się różnie – brakiem odpowiedniego zapisu w kontrakcie z klientem, wewnętrznymi procedurami bądź brakiem czasu na zorganizowanie pracownikom klienta dostępu do środowiska testowego. Jednak z punktu widzenia klienta utrudnianie wglądu w wyniki i przebieg testów wynika z próby ukrycia niskiej jakości testowania – co wychodzi na jaw przy oficjalnym odbiorze systemu czy przejściu do etapu testów akceptacyjnych.

Tymczasem udostępnienie klientom środowiska testowego nie tylko pokazuje, że dostawca nie ma nic do ukrycia i rzetelnie wykonuje testy zewnętrzne, ale i może wspomóc procesy QA poprzez wiedzę i doświadczenie pracowników klienta. Z punktu widzenia klienta, wgląd w wewnętrzne procesy testowe jest niezbędny dla upewnienia się, że jakość aplikacji będzie zgodna z deklaracjami i wszystkie krytyczne obszary systemu zostaną zweryfikowane na odpowiednim poziomie. Aktualna informacja o kondycji systemu umożliwia nie tylko lepsze planowanie dalszej pracy zespołu klienta (tzn. przygotowanie do testów akceptacyjnych), ale i wspólne opracowanie odpowiednich mechanizmów korygujących na wypadek pojawienia się problemów podczas implementacji czy testowania.

Nie tylko wyniki testów mają znaczenie – często klient pragnie również zweryfikować strategię realizacji testów wewnętrznych celem upewnienia się, że umożliwi weryfikację oprogramowania w wystarczającym stopniu. Pracownicy klienta – jako osoby (z reguły) pracujące w danej branży i doskonale znające dany biznes – są w stanie pomóc w ocenie, czy proponowane podejście i technika sprawdzą się do weryfikacji określonych funkcji systemu.

Często klient oczekuje również wglądu w przypadki testowe zaprojektowane do testów wewnętrznych – zazwyczaj systemowych. Zdarza się, że dostawca IT odbiera to jako próbę kontroli i ingerencji w wewnętrzne procesy, lub wręcz dodatkową pracę dla swoich pracowników, którzy muszą dostosować treść i zakres dokumentacji testowej do możliwości i oczekiwań klienta. Tymczasem celem klienta nie jest utrudnianie pracy innym, a raczej usprawnienie testowania poprzez upewnienie się, że aplikacja będzie testowana w odpowiedni sposób, oraz że wszystkie krytyczne obszary i funkcjonalności zostaną zweryfikowane. Podczas takiego klienckiego przeglądu przypadków testowych można często wykryć braki w projekcie testów i uzupełnić je przed rozpoczęciem prac QA lub już w ich trakcie (lecz przed oddaniem aplikacji do klienta). Inną korzyścią z przeglądu jest to, że klient może sprawdzić, jakie testy wykonuje dostawca, i wyłączyć owe testy z zakresu swoich prac – w zamian projektując inne, uzupełniające. Umożliwia to nie tylko wyeliminowanie ryzyka dublowania pracy, ale i rozszerza zakres i głębokość testów.

Odbiór systemu

Problemy pojawiają się również przy odbiorze systemu od dostawcy (przekazaniu do testów klientowi). Zdarzają się braki zarówno w dokumentacji – specyfikacjach, raportach z testów – jak i w samym systemie – błędy w implementacji, i to w funkcjonalnościach deklarowanych jako działające poprawnie. Klient nierzadko ma wrażenie, że dostawca nie wykonywał testów rzetelnie i zgodnie z wymaganiami jakościowymi – czyli nie wywiązał się z umowy dotyczącej jakości produkowanego systemu. W takiej sytuacji klient nie wie, jakie testy zostały rzeczywiście wykonane, jakie obszary zweryfikowane i w jakim stopniu oraz na jakich testach powinien się skupić – czy też raczej sprawdzać wszystko od początku, polegając na własnej ocenie jakości systemu.

Jakie problemy mogą pojawić się przy – lub już po – odbiorze systemu?

- Dostawca nie przekazuje całości dokumentacji potrzebnej do prawidłowego użytkowania czy dalszych testów systemu (głównie specyfikacje i dokumentacje użytkownika oraz końcowe raporty z testów). Może to wynikać zarówno z niedopatrzenia, jak i braku takiej dokumentacji u dostawcy. Jeżeli umowa o wykonanie oprogramowania jasno określa elementy i produkty, które będą musiały zostać dostarczone wraz z systemem, sytuacja taka nie powinna w ogóle mieć miejsca, jako że jest to naruszenie postanowień kontraktu i niewywiązanie się z niego.
- Dostawca nie weryfikuje jakości oprogramowania po jego instalacji w środowisku docelowym. Producent powinien upewnić się, iż aplikacja prawidłowo startuje i funkcjonuje po przeniesieniu ze środowiska deweloperskiego czy testowego na środowisko klienta i dopiero po tym oficjalnie przekazać system. Dlaczego? Dlatego, iż zdarza się, że z różnych powodów aplikacja działająca w środowisku testowym czy deweloperskim nie działa u klienta – czy to przez różnice w konfiguracji środowisk, czy też przez zwykłe błędy, które wystąpiły podczas instalacji systemu.
- Jakość deklarowana przez dostawcę nie odpowiada faktycznemu stanowi rzeczy. W trakcie testów klienta czy też użytkowania systemu na produkcji może okazać się, że funkcjonalności deklarowane jako przetestowane i działające w 100% zawierają błędy uniemożliwiające korzystanie z aplikacji zgodnie z jej celem i przeznaczeniem. Przypadki i scenariusze testowe, w raporcie z testów określone jako zrealizowane z wynikiem pozytywnym, w użyciu powodują mniej lub bardziej krytyczne błędy i nie realizują swoich funkcji, wprawiając testerów czy użytkowników w irytację i niwecząc dotychczasową koncepcję pracy z systemem. W takiej sytuacji pracownicy klienta nie wiedzą, czego mogą się spodziewać po zachowaniu oprogramowania i przestają wierzyć zapewnieniom dostawcy.
- Dostawca nie wspiera testowania i działań serwisowych w oczekiwanym stopniu. Jeśli ustalenia zawarte w kontrakcie z dostawcą nie są wystarczająco precyzyjne i nie określają jasno zakresu odpowiedzialności i prac serwisowych oraz wspierających inne prace, klient może spotkać się z problemami w tym temacie. Dostawca może nie chcieć udostępniać swoich dotychczasowych wyników testów, weryfikować podejścia i strategii do testowania klienta czy też realizować zgłoszenia błędów pochodzące od zespołu klienta bez należytej staranności. Ta ostatnia kwestia może stanowić duży problem, a niestety zdarza się stosunkowo często – dostawca nie przyjmuje zgłoszeń odpowiednio szybko, zwleka z naprawianiem błędów czy nie sprawdza poprawności naprawionych funkcji czy modułów przez przekazaniem ich klientowi.

Co zrobić, by uniknąć problemów?

Podstawą zapewnienia odpowiedniej jakości procesu wytwarzania systemu jest kontrakt zawierany z dostawcą IT i zawarte w nim zapisy i wymagania. Zainteresowanie kwestiami jakości procesu i produktów projektu dopiero na etapie implementacji, testowania czy – jeszcze gorzej – wdrażania systemu na produkcję, to stanowczo za późno. Na takim etapie klient może mieć trudności przy wyegzekwowaniu pewnych działań (korygujących czy zapobiegających), szczególnie jeśli nie były one przewidziane oraz udokumentowane w kontrakcie w ramach zakresu prac realizowanych przez dostawcę.

Jak wynika z informacji udzielanych przez pracowników działów IT organizacji klientów, bardzo często zdarza się, że umowy konstruowane są niemalże ad hoc, bez należytej staranności we wszystkich istotnych kwestiach. Wynika to najczęściej z braku wiedzy ludzi odpowiedzialnych za tworzenie umów – nie zawsze pracownicy klienta zdają sobie sprawę, jak dokład-

Czego oczekuje klient?

Zwykle klient nie ingeruje w wewnętrzny proces dostawcy i nie narzuca własnych rozwiązań. Z reguły klient oczekuje jedynie:

- wglądu w dokumentację testową, a w szczególności:
- w plan testów;
- w specyfikacje przypadków użycia;
- w dane testowe.
- udostępnienia wyników testów w określonym zakresie (np. dla testów krytycznych funkcjonalności);
- informacji o problemach pojawiających się podczas testowania (opóźnieniach wynikających z implementacji wielu zmian, niedostępności danych czy środowisk testowych itp.).

Odbiór systemu

Co powinno być dostarczone przy przekazaniu systemu? Co najmniej następujące elementy:

- *Specyfikacja systemu* (łącznie z dokumentacją użytkownika) – aktualna dokumentacja systemu jest konieczna, szczególnie w przypadku, gdy klient nie brał udziału w testach systemowych i przystępuje od razu do testów akceptacyjnych. Mimo iż przypadki testowe do UAT są zwykle projektowane na wczesnym etapie testów, aktualna dokumentacja systemu służy jako podstawa ewentualnej aktualizacji testów (np. gdy zmienność aplikacji była stosunkowo duża) oraz dodatkowe źródło informacji o funkcjonalności oprogramowania.
- *Raporty końcowe z testów* – dostarczają informacji o stopniu, w jakim aplikacja została przetestowana, jakie pojawiały się błędy i ile ich było (co umożliwia określenie obszarów zwiększonego ryzyka). Zwykle raport końcowy z testów zawiera:
  - wstęp;
  - streszczenie – omówienie i wyniki oraz obserwacje, ewentualnie zalecenia;
  - środowisko i metodyka testowania;
  - wyniki testów;
  - zgłoszone problemy.
- *Lista otwartych błędów* – lista błędów, o których dostawca wie, jest niezbędnie wymagana przy przekazaniu systemu do klienta. Dzięki temu klient zdaje sobie sprawę, że aplikacja zawiera określone błędy i może nie działać w pełni zgodnie ze specyfikacją; wie również, że dostawca jest świadomy istnienia owych błędów i zamierza je naprawić. Jawna lista otwartych błędów eliminuje sporo problemów związanych z testowaniem, które będzie realizował zespół klienta – wiadomo, co działa i co może być testowane, gdzie pojawi się błąd oraz czy znaleziony w trakcie pracy defekt jest znany dostawcy, czy też nowy.

nie przebiega proces wytwarzania oprogramowania i o co należy zadbać na etapie uzgodnień warunków umowy. W efekcie w kontrakcie brakuje np. wyszczególnienia wszystkich prac, które dostawca zobowiązuje się wykonać oraz określenia, w jakiej postaci klient oczekuje dowodów realizacji owych prac.

Czy konieczne jest zatem zatrudnienie specjalisty na stanowisko osoby odpowiadającej za opracowywanie kontraktów? Nie zawsze. Jeśli zlecenie wykonania usług IT jest jednorazowe, lub wszystko wskazuje na to, iż dana organizacja nie będzie zbyt często korzystać z usług zewnętrznych dostawców IT, nie jest ani konieczne, ani opłacalne utrzymywanie takiego stanowiska. W przypadku, gdy klient nie posiada personelu o odpowiednich kompetencjach, może zawsze zaangażować zewnętrznych konsultantów, którzy będą prowadzić negocjacje z dostawcą i zadbają o odpowiednią konstrukcję umowy. Rola konsultanta może nie ograniczać się do nadzoru nad zapisami umowy; konsultanci często uczestniczą we wszystkich pracach projektowych, a ich zadaniem jest upewnienie się, że dostawca wypełnia warunki umowy oraz wspieranie organizacji klienta w obszarach, w których pracownicy klienta nie mają wystarczających kompetencji. Przykładowo – konsultanci mogą dokonywać przeglądów dokumentacji analitycznej, by zweryfikować jej poprawność i zgodność z wymaganiami, jak również np. zgodność ze standardami jakości wymaganymi przez daną organizację. Dość często konsultanci z zewnętrznych instytucji zatrudniani są w charakterze analityków biznesowych albo managerów testów, wspierając klienta w pracach wymagających specyficznej wiedzy i doświadczenia.

Tabela 1. *Elementy zapewnienia jakości*

Faza	Środek	Lista zagadnień
Start projektu	Umowa	Wykaz prac realizowanych przez dostawcę
		Wykaz produktów poszczególnych faz i etapów
		Kryteria akceptacji / odbioru produktów
		Konsekwencje na wypadek nie spełnienia wymagań
		Czasy reakcji na problemy
Implementacja i testowanie	Plan testów	Strategia i podejście do testów
		Harmonogram
		Środowisko testowe
		Narzędzia
		Punkty kontrolne
	Projekt testów	Lista przypadków testowych
		Pokrycie wymagań testami
	Raport z testów	Wyniki testów
		Liczba otwartych błędów
		Statystyki defektów (identyfikacja obszarów o zwiększonym ryzyku)
Przekazanie systemu	Dokument odbioru systemu	Jak plan testów i raport z testów
		Lista zmian

Tabela 2. *Wykaz czynności i produktów faz projektu*

Faza	Czynność	Produkt
Analiza biznesowa	Określenie szczegółowych wymagań biznesowych	Lista wymagań
	Określenie reguł dotyczących wyglądu aplikacji	Dokument *Look & Feel*
	Udokumentowanie wymagań	Specyfikacja funkcjonalna (diagramy przypadków użycia, modele procesów)
Projektowanie	Określenie architektury systemu	Dokument architektury
	Analiza integracji z systemami zewnętrznymi, określenie mapowania danych	Dokumentacja mapowania
	Stworzenie struktury danych	Diagramy przepływu danych Słownik danych
Testowanie	Opracowanie strategii testów	Plan testów
	Zaprojektowanie przypadków testowych	Projekt testów (w tym szczegółowe przypadki testowe)
	Wykonanie testów	Raporty błędów
	Analiza wyników testów i opracowanie raportów końcowych z testów	Raport końcowy
Wdrożenie	Instalacja systemu w środowisku docelowym	Dokumentacja instalacji i wdrożenia
	Szkolenie użytkowników systemu	Podręcznik użytkownika

Wracając do samej umowy o realizację usług informatycznych – klient powinien upewnić się, że treść umowy obejmuje następujące elementy:

Zakres prac: wyszczególnienie etapów projektu IT wraz z wykazem głównych prac realizowanych w ramach każdego etapu. Umowa powinna jasno określać główne fazy projektu oraz najważniejsze czynności (przykład w Tabeli 2).

Produkty projektu: główne (czyli sam system informatyczny wraz z dokumentacją systemu) oraz pośrednie (dokumentacja analityczna, projektowa, testowa, wdrożeniowa). Produkty powinny być określone już na etapie kontraktu, aby od początku było jasne, co dostawca musi wykonać i kiedy.

Kryteria akceptacji produktów projektu – klient powinien określić warunki, jakie muszą być spełnione, by dany produkt mógł być zaakceptowany jako spełniający określone kryteria jakościowe. W przypadku dokumentacji będzie to rzecz jasna przejrzystość, poprawność, kompletność, jak również zgodność z wymaganiami systemu oraz standardami jakości (np. ze standardem)

Wymagania dotyczące procesu produkcji oprogramowania (np. zgodność ze standardem ISO/IEC 12207, IEEE-Std-601). Klient ma prawo wymagać od dostawcy realizowania projektu według określonej metodologii czy żądać, by proces był zgodny ze standardami – nie jest tak, że to dostawca narzuca swój własny proces, według którego postępuje zawsze. Oczywiście nie należy na siłę ustalać danej metodologii (bo np. jest modna), lecz przeanalizować, jakie podejście najlepiej pasuje do projektu i daje największe szanse powodzenia (w tym momencie często przydaje się specjalista – konsultant z innej organizacji).

Wymagania dotyczące zespołu produkcyjnego – w niektórych przypadkach wskazane jest określić dokładne wymagania związane z niezbędnymi kwalifikacjami czy doświadczeniem członków zespołu projektowego – aby na wstępie wyeliminować ryzyko, iż dostawca zaangażuje do prac projektowych personel o zbyt niskich kompetencjach. Dla przykładu – w projekcie mającym na celu wyprodukowanie złożonego systemu bankowego powinny uczestniczyć osoby, które posiadają kilkuletnie doświadczenie w tego typu projektach – ponieważ realizacja takiego projektu wiąże się z koniecznością posiadania specjalistycznej wiedzy w danej dziedzinie i znajomości branży. Z reguły sam dostawca dba o zapewnienie personelu o odpowiednich kompetencjach, jako że powodzenie projektu leży w jego interesie – warto jednak również ze strony klienta upewnić się, iż do zespołu nie trafią osoby nie spełniające pewnych wymagań: w tym celu w umowie powinien znaleźć się odpowiedni zapis (zwykle formułowany w postaci wymagania: Dostawca zapewnia dostępność personelu ... po czym następuje wyszczególnienie ról projektowych i skróconego opisu niezbędnych kwalifikacji).

Terminy wykonania prac, w tym również poprawek – często jest to pomijane lub nie dość precyzyjne, np. klient nie określa terminów, w jakich dostawca powinien naprawiać błędy o określonym priorytecie... i w efekcie jest tak, że otwarte błędy wiszą tygodniami, ponieważ nie określono tak zwanych czasów reakcji i naprawy defektów. O ile nie ma to zbyt wielkiego znaczenia w przypadku błędów kosmetycznych, o tyle jest krytyczne dla np. postępu prac testowych, jeśli błędy są poważne i blokują część funkcjonalności.

Sam kontrakt nie zapewni, że dostawca prawidłowo wywiąże się ze swoich zobowiązań. Klient powinien zapewnić sobie i inne środki – wgląd w wyniki testów czy nawet wsparcie testowania to jedna z metod. Klient powinien wymagać od dostawcy przekazania co najmniej następującej dokumentacji:

- *plan testów* – przed rozpoczęciem testowania – tak, by klient mógł odnieść się do podejścia proponowanego przez dostawcę i ewentualnie wprowadzić swoje sugestie (przykładowo pomóc w określeniu ryzyka);
- *raport z testów* – po zakończeniu testów.

Dobrą praktyką jest zapewnienie sobie wglądu w przypadki i scenariusze testowe – przynajmniej dla testów systemowych lub w ramach dostarczenia dowodów spełnienia wymagań systemowych.

Plan testów, oprócz harmonogramów i podziału prac, powinien opisywać również ogólne podejście (strategię) testowania – wybraną metodę wraz z jej uzasadnieniem, zidentyfikowane poziomy testów, wyszczególnienie cykli (iteracji), o ile jest to możliwe na etapie planowania, oraz kryteriów wejścia i wyjścia dla poszczególnych poziomów. Zdarza się, iż dostawca odmawia poinformowania klienta o realizowanej przez siebie strategii testowania i metod wykorzystywanych do weryfikacji oprogramowania na różnych etapach, tłumacząc się tym, że ta wiedza jest jego *know-how* – tajemnicą firmową, i nie udostępnia jej inaczej, niż poprzez pełnopłatne, zorganizowane szkolenia.

Raport z testów musi zawierać listę wykonanych przypadków testowych wraz z ich rezultatami oraz statystyki defektów – znalezionych, naprawionych, zamkniętych w poszczególnych sesjach testowych. Zwykle w raporcie końcowym umieszcza się również wykaz otwartych (nie naprawionych w finalnej bądź bieżącej wersji systemu) błędów oraz potencjalnych problemów (np. system nie wspiera określonych modeli urządzeń, systemów operacyjnych, wersji programów) i rekomendacji.

Przypadki i scenariusze testowe zwykle nie są oficjalnie wymaganą dokumentacją przy akceptacji testów czy też odbiorze systemu. Mogą one być oczywiście udostępniane klientowi – o ile jest taki zapis w umowie lub uzgodnienia pomiędzy stronami – na określonym etapie testów lub jako dowód spełnienia wymagań systemu (zwłaszcza w podejściu do testowania opartym na wymaganiach). Jednak niezależnie od tego, czy klient wymaga okazywania mu dokumentacji przypadków testowych na poszczególnych etapach testowania, czy też nie, dostawca powinien takową dokumentację opracować i utrzymywać – chociażby dla kontroli postępu prac testowych i stopnia realizacji wymagań. W przypadku, gdy dostawca stanowczo odmawia klientowi wglądu w projekt testów, może to oznaczać, iż dokumentacji testowej po prostu nie ma, a prace QA wykonywane są w sposób nieuporządkowany, bez planu i mechanizmów nadzoru.

Kolejną metodą zapewnienia prawidłowego wykonywania prac przez dostawcę jest tak zwany dokument odbioru systemu, wymagany od dostawcy przy przekazaniu systemu z testów wewnętrznych do zespołu klienta, który kontynuuje testowanie zgodnie z własnym podejściem i rozumieniem idei systemu.

Dokument odbioru systemu opisuje proces testowy dostawcy – podejście do testowania,

W czym może pomóc konsultant z zewnątrz?

- Negocjacje warunków umowy, zakresu projektu oraz wymagań.
- Zapewnienie prawidłowej konstrukcji umowy z dostawcą IT tak, by przejrzyście i wystarczająco szczegółowo opisywała zakres poszczególnych prac, jakie dostawca zobowiązuje się wykonać w ramach danego projektu oraz kryteria akceptacji/odbioru produktów projektu.
- Zidentyfikowanie obszarów ryzyka i zapewnienie, by w kontrakcie znalazły się zapisy o konsekwencjach, jakie poniesie dostawca w razie niewywiązania się z warunków umowy.
- Wsparcie w wyborze odpowiedniej dla danego projektu metody/podejścia do zarządzania, analizy, projektowania, testowania (szczególnie przydatne w przypadku, gdy klient uczestniczy w wytwarzaniu produktu na każdym etapie projektowym)/
- Identyfikacja szczegółowych wymagań systemowych i przekazanie ich dostawcy w określonej formie.
- Recenzowanie dokumentacji projektowej celem weryfikacji jej poprawności, zgodności ze standardami czy praktykami, stopnia spełnienia wymagań.
- Wsparcie procesu testowego – zarządzanie testami ze strony klienta, nadzór/kontrola realizacji testów wykonywanych przez dostawcę, weryfikacja rezultatów testów, analiza pokrycia wymagań testami, identyfikacja danych testowych etc.

Dokument odbioru systemu

1.Testy
2.Testowane produkty i komponenty
2.1.Funkcjonalności podlegające testom
2.2.Funkcjonalności nie podlegające testom
2.3.Podejście do testów
2.3.1.Typy testów
2.3.2.Techniki testowania
2.3.3.Narzędzia do zarządzania i wykonywania testów
2.3.4.Kryteria zakończenia testów
2.4.Testy regresji
1.Harmonogram
2.Zespół testowy
3.Środowisko testowe
5.1.Parametry środowiska testowego
5.2.Dane testowe
1.Wyniki testów
6.1.Stan przypadków testowych
6.1.1.Wersja x.x systemu
6.1.1.1.Pozytywne
6.1.1.2.Negatywne
6.1.1.3.Niewykonane
6.1.1.4.Zablokowane
6.2.Pokrycie wymagań testami
6.3.Otwarte defekty
6.4.Statystyki defektów
6.5.Lista zmian
6.5.1.Zmiany zrealizowane
6.5.2.Zgłoszone zmiany

założenia, wyniki testów, statystyki defektów w poszczególnych cyklach testowych. Jaki jest cel tworzenia takiego dokumentu?

Po pierwsze, wyjaśnienie, w jaki sposób dostawca realizował testy, w jakich warunkach (np. konfiguracja środowiska) i jakie uzyskał wyniki. Na podstawie tych danych klient może opracować plan testów, jaki sam będzie wykonywał, uwzględniając w szczególności obszary niestabilne i o zwiększonym ryzyku (czyli te, w których znajdowano najwięcej błędów).

Po drugie, dostarczenie dowodów, że dostawca wypełnił wymagania umowy w zakresie weryfikacji systemu (np. maksymalna liczba otwartych błędów o wysokim priorytecie na zakończenie testów to 2, w tym żaden nie blokuje głównych funkcjonalności). Dokument odbioru systemu jest w pewnym sensie deklaracją jakości – dostawca stwierdza, iż wykonał takie a takie testy w określonym zakresie i uzyskał dane wyniki. Jeśli w trakcie testów klienta okaże się, że przedstawione dane nie odpowiadają prawdzie (np. jakaś funkcjonalność nie została przetestowana, co wynika z dużej liczby błędów odkrytych przez klienta), klient może pociągnąć dostawcę do odpowiedzialności. Należy tu jednak mieć na uwadze, że niektóre błędy pojawiają się dopiero w środowisku klienta (co wynika z różnic w konfiguracji, bazie danych etc.) i dostawca nie miał możliwości, by je znaleźć. Dlatego też elementem dokumentu odbioru systemu jest punkt środowisko testowe, który opisuje parametry środowiska testowego, w którym dostawca wykonywał swoje prace.

Po trzecie, zapewnienie wykonania pewnych prac w ramach testów – na przykład testów regresji po każdej zmianie jakiejś części systemu lub naprawie błędu wpływającego na wiele funkcjonalności. Mimo że testy regresji wydają się rzeczą oczywistą, nie zawsze tak jest – określona funkcjonalność, która podlegała dużym zmianom działa błędnie, za to inne części systemu szwankują, ponieważ zmiany wywołały nowe błędy, a dostawca zapomniał przetestować system pod tym kątem.

Podsumowanie

Podpisując kontrakt o świadczenie usług IT, dostawca zobowiązuje się wykonać określone prace w określonym terminie i otrzymując odpowiedni poziom jakości. Klient może wierzyć na słowo w deklaracje spełnienia owych wymagań (i załamywać ręce, kiedy okazuje się, że deklaracje mijają się z faktami) – może też zapewnić sobie pewne mechanizmy kontroli i nadzoru nad poprawnością wykonywanych prac i ich wynikami. Postawienie dostawcy określonych warunków nie jest niczym więcej niż tylko zapewnieniem, iż w ramach wynagrodzenia za wykonanie usługi dostawca spełni oczekiwania klienta i dostarczy produkt zgodny z wymaganiami, w ustalonym terminie oraz o określonym poziomie jakości. Jeśli zakup książki w sklepie internetowym pozwala klientowi wybrać dostawcę oferującego najkorzystniejszą cenę, określić warunki dostawy i możliwość zwrotu towaru w przypadku, gdy produkt nie odpowiada oczekiwaniom – tym bardziej należy stosować podobne zasady w przypadku oprogramowania, którego zakup jest drogi i związany z wysokim ryzykiem.

Niniejszy artykuł przybliżył podstawowe kwestie, o których powinien pamiętać klient, przystępując do współpracy z dostawcą IT oraz przedstawił najprostsze sposoby zapewnienia prawidłowego wytwarzania produktu. Opisane metody w większości nie wymagają szczególnego nakładu czasu i środków, a mogą przynieść wymierne korzyści. Jasne postawienie warunków współpracy na początku minimalizuje ryzyko nieporozumień i konfliktów w przyszłości, a w dłuższym czasie pozwala obu stronom czuć się bardziej komfortowo – dostawcy, ponieważ wie dokładnie, czego się od niego oczekuje, i klientowi – jako że otrzyma obiektywne dowody na to, że kontrahent wypełnił nałożone na niego obowiązki.

W Sieci

- *http://iir.nf.pl/* – strona firmy szkoleniowej, mającej w ofercie m.in. szkolenia dotyczące optymalnego konstruowania umów z dostawcami zewnętrznymi IT w sposób zapewniający spełnianie gwarancji zawartych w umowach SLA, poprawiający efektywność kosztową świadczonych usług i pozwalający na budowanie wysokiej relacji między dostawcą a klientem;
- *www.outsourcing.com.pl/559,jak_podpisac_dobra_umowe_na_outsourcing.html* – artykuł na temat konstruowania umów z dostawcami IT – sporo wskazówek i cennych informacji na temat niezbędnych elementów umowy, w tym aspektów kontroli i ceny;
- *biznes.gazetaprawna.pl/artykuly/11516,dostawca_it_odpowiada_za_oprogramowanie.html* – artykuł z Gazety Prawnej, opisujący m.in. jakie praktyki dostawca może pokryć pod pojęciem klauzuli umownej;
- *mojafirma.infor.pl/poradniki/449,2542,5,.html* – obszerny artykuł na temat prawnych aspektów umów wdrożeniowych, oraz wiele wskazówek, jakie informacje powinny znaleźć się w treści umowy.

Pojęcia

- *Analiza pokrycia wymagań testami* – analiza mająca na celu stwierdzenie, w jakim stopniu zaprojektowane przypadki testowe obejmują (pokrywają) określone wymagania (funkcjonalne i niefunkcjonalne);
- *Czas reakcji* – termin, w jakim dostawca IT zobowiązany jest podjąć działania w odpowiedzi na zgłoszony problem;
- *Czas naprawy* – termin, w jakim dostawca IT zobowiązany jest dostarczyć rozwiązanie zgłoszonego problemy do użytkowników;
- *Testy akceptacyjne* (ang. *acceptance testing*) – celem jest sprawdzenie, czy oprogramowanie jest gotowe i może być przekazane użytkownikowi końcowemu;
- *Testy ALFA* – wykonywane w środowisku wytwórczym przez wytwórców oraz klientów, aby sprawdzić, czy stworzony system jest akceptowalną implementacją wymagań;
- *Testy BETA* – wymagają dostarczenia systemu do pewnej liczby użytkowników końcowych, którzy zgodzili się wykorzystywać system i przekazywać pojawiające się problemy producentowi;
- *Testy integracyjne* (ang. *integration testing*) – wykonywane w celu sprawdzenia, czy komponenty współpracują ze sobą poprawnie np. podczas realizacji przypadku użycia; służy do wychwycenia niekompletności lub błędów w specyfikacjach interfejsów pakietów.
- *Testowanie regresyjne* (ang. *regression testing*) – wykonywane po wprowadzeniu zmian do oprogramowania w celu zapewnienia, że nowe defekty nie zostały wprowadzone jako rezultat zmian; ten typ testów porównuje aktualną wersję systemu z poprzednią wersją i identyfikuje różnice jako potencjalne defekty (przy założeniu, że zachowanie nie powinno ulec zmianie);
- *Testy systemu* (ang. *system testing*) – rozpoczynają się, gdy oprogramowanie jako całość lub jego jasno określona część jest funkcjonalna (sprawna).

KAROLINA ZMITROWICZ

Pracuje na stanowisku Analityka biznesowego w firmie Profi-Data. Karolina specjalizuje się obecnie w modelowaniu wymagań biznesowych. Wcześniej pracowała jako Manager Quality Assurance w projektach informatycznych w sektorze finansowo – bankowym.

Profi-Data to producent oprogramowania specjalizujący się w budowie dedykowanych rozwiązań informatycznych dla ubezpieczeń, bankowości i administracji publicznej. Firma posiada certyfikat jakości ISO 9001:2000 w zakresie produkcji oprogramowania.

Kontakt z autorem: karolina_zmitrowicz@wp.pl

Jakbyś tam był

muzyka.wp.pl

Wejdź na muzyka.wp.pl / czytaj relacje z koncertów
/ poczuj prawdziwe emocje / przeglądaj recenzje
/ daj się porwać koncertowej atmosferze / poczuj to, jakbyś tam był

WP.PL

Dziedziczenie

Tworzenie hierarchii klas

W poprzednim odcinku Akademii UML poznaliśmy sytuacje, w których nie należy stosować dziedziczenia. Zobaczmy, kiedy powinniśmy tworzyć wspólną nadklasę dla kilku klas, a kiedy taka hierarchia klas jest zbędna.

Dowiesz się:
- Kiedy należy stosować dziedziczenie;
- Jak unikać nadmiernej hierarchii klas.

Powinieneś wiedzieć:
- Znać notację diagramów klas języka UML;
- Znać podstawowe zasady analizy i projektowania obiektowego;
- Rozumieć mechanizm dziedziczenia.

Poziom trudności

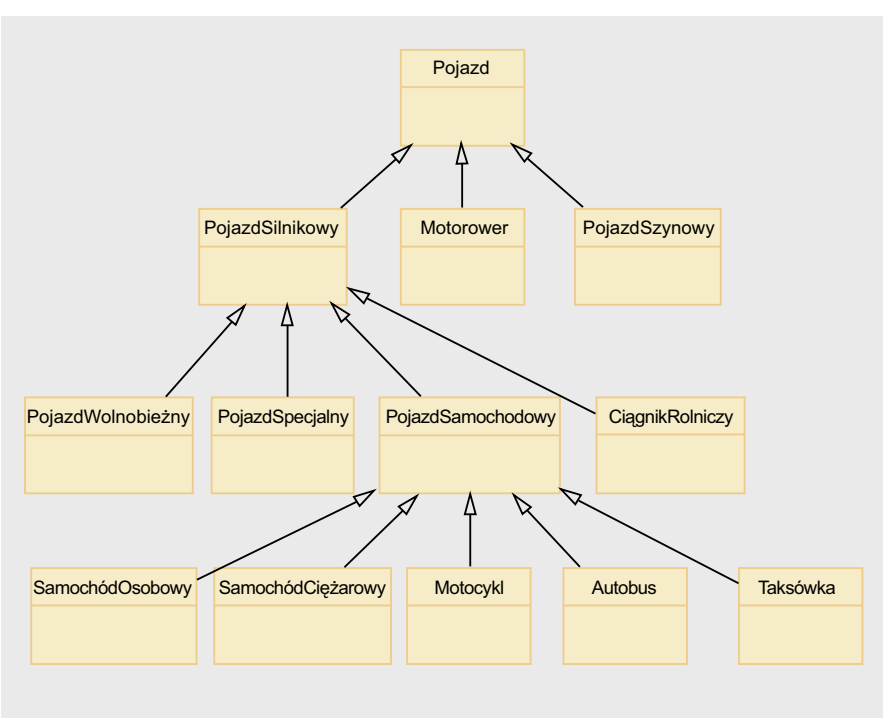
Wyobraźmy sobie, że projektujemy system dla instytucji ubezpieczeniowej, obsługujący ubezpieczenia komunikacyjne pojazdów. Podczas analizy dowiadujemy się, że system ten ma w szczególności działać zgodnie z ustawą *Prawo o ruchu drogowym*. Tworząc diagram klas dla naszego systemu, musimy w nim oczywiście umieścić klasy reprezentujące pojazdy, które będą podlegały ubezpieczeniom. I tu przychodzi nam z pomocą wspomniana ustawa, w której znajdziemy precyzyjne definicje poszczególnych rodzajów pojazdów, np.:

Pojazd samochodowy – pojazd silnikowy, którego konstrukcja umożliwia jazdę z prędkością przekraczającą 25 km/h; określenie to nie obejmuje ciągnika rolniczego

Na podstawie takich definicji moglibyśmy stworzyć hierarchię klas taką jak na Rysunku 1.

Model ten jest najzupełniej poprawny i nie można mu niczego zarzucić. Jest to jednak znakomity przykład pułapki, w jaką można wpaść, modelując rzeczywistość biznesową. Pułapka ta polega na próbie zamodelowania wszystkich możliwych aspektów rzeczywistości, także tych, które są nieistotne z punktu widzenia projektowanego systemu. Tymczasem cała sztuka polega na tym, aby *oddzielić ziarno od plew*, czyli umieścić w modelu aspekty istotne dla naszego zastosowania, pomijając rzeczy nieistotne.

W tę pułapkę szczególnie łatwo jest wpaść, korzystając z dziedziczenia i tworząc hierarchię klas, która wykracza poza zakres projektowanego systemu. Jeśli założymy, że nasza instytucja ubezpieczeniowa zajmuje się przede wszystkim ubezpieczeniami samochodów osobowych, samochodów ciężarowych oraz motocykli, to klasy takie jak `PojazdWolnobieżny` czy `PojazdSzynowy` są zbędne, ponieważ wykraczają poza dziedzinę nasze-



Rysunek 1. *Hierarchia klas pojazdów*

Object Constraint Language (OCL)
OCL jest językiem wyrażeń, umożliwiającym formułowanie ograniczeń (ang. *constraints*) dla modeli obiektowych i innych artefaktów powstałych w czasie modelowania obiektowego. (J. Warmer, A. Kleppe, *OCL – precyzyjne modelowanie w UML*, WNT, 2003)
OCL jest częścią języka UML. Specyfikacja OCL została dodana do UML w wersji 1.1.

go systemu. Niepotrzebne są także klasy `PojazdSilnikowy` i `Pojazd`, ponieważ nie mają żadnych atrybutów ani nie są powiązane z innymi klasami, a więc jedynym ich zadaniem jest grupowanie podklas. Umieszczanie w modelu takich klas to zwykłe jego zaśmiecanie.

Nawet jeśli sporadycznie zdarza się naszej instytucji ubezpieczać np. pojazdy szynowe czy specjalne, to zapewne wystarczy wrzucić je do jednego worka, tworząc klasę `InnyPojazd` i przyjmując, że obiekty tej klasy będą ubezpieczane na zasadach niestandardowych (w przeciwieństwie do np. samochodów osobowych, dla których obowiązuje pewna ustandaryzowana oferta). W ten sposób otrzymamy model przedstawiony na Rysunku 2 – bardziej zwięzły, niezaśmiecony klasami nie należącymi do dziedziny systemu.

Klasy (pozornie) ze sobą niezwiązane

Nie zawsze jednak unikanie nadklas jest dobrym rozwiązaniem. Czasami warto stworzyć wspólną nadklasę dla klas, które pozornie nie mają ze sobą nic wspólnego.

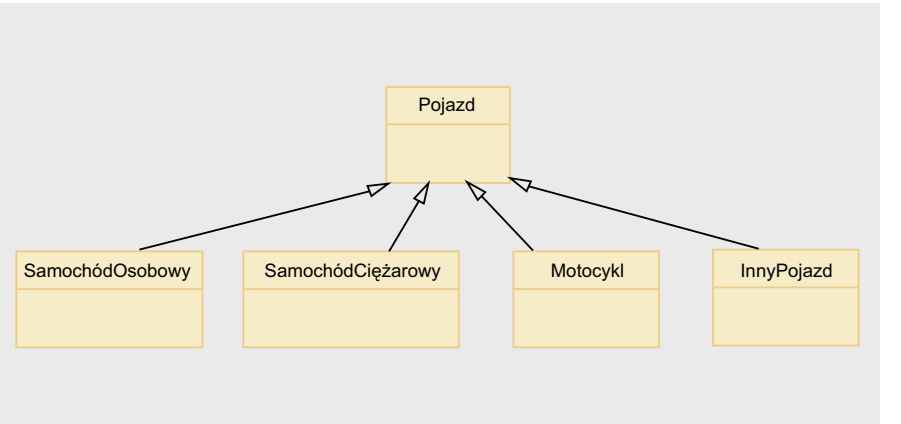
Wyobraźmy sobie system wspierający naliczanie i wypłaty dopłat dla rolników. Każdy rolnik objęty dopłatami posiada zwierzęta oraz działki rolne (aby być rolnikiem, musi posiadać co najmniej jedno zwierzę lub działkę rolną). Co roku za każde posiadane zwierzę oraz każdą działkę rolną rolnikowi przysługuje dopłata. Załóżmy dla uproszczenia, że kwota dopłaty za zwierzę zależy od jego wagi, zaś kwota dopłaty za działkę – od jej powierzchni. Aby system mógł naliczać dopłaty, musimy w nim przechowywać dane wszystkich rolników oraz posiadanych przez nich zwierząt i działek. Moglibyśmy to robić zgodnie z modelem przedstawionym na Rysunku 3.

Model ten ma jedną wadę – dopuszcza istnienie rolników, którzy nie posiadają ani działek, ani zwierząt, co jest niezgodne z podaną definicją rolnika. Tego ograniczenia nie da się niestety zapisać przy pomocy liczebności (`1..*` zamiast `0..*`), bo to oznaczałoby, że każdy rolnik musi posiadać co najmniej jedno zwierzę i co najmniej jedną działkę. W takiej sytuacji zawsze możemy poradzić sobie, zapisując (poza samym diagramem) dodatkową regułę biznesową. Regułę taką zapisujemy najczęściej słownie (*rolnik musi posiadać co najmniej jedno zwierzę lub co najmniej jedną działkę*) lub w postaci wyrażenia (niezmiennika – ang. *invariant*) w języku OCL dla klasy `Rolnik`:

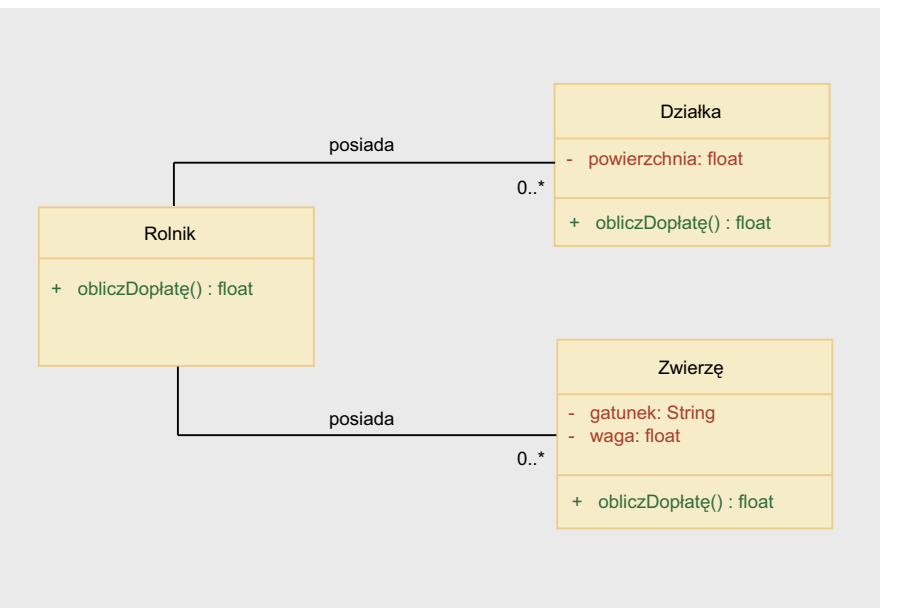
```
Działka->size + Zwierzę->size >= 1
```

Zamiast pisać regułę biznesową, warto jednak pomyśleć o modyfikacji samego modelu.

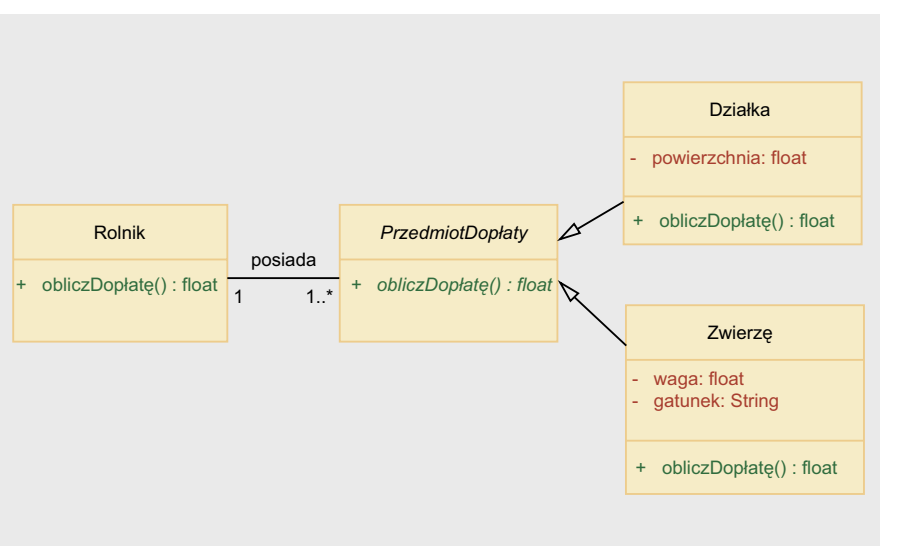
Na pierwszy rzut oka wydaje się, że klasy `Działka` i `Zwierzę` reprezentują zupełnie różne byty, nie mające ze sobą nic



Rysunek 2. *Uproszczona hierarchia klas pojazdów*



Rysunek 3. *Model systemu naliczania dopłat – bez użycia dziedziczenia*



Rysunek 4. *Model systemu naliczania dopłat – z użyciem dziedziczenia*

wspólnego. Faktycznie, kawałek ziemi uprawnej rzadko kiedy przypomina zwierzę hodowlane, więc trudno byłoby znaleźć dla nich wspólną nadklasę. Czy rzeczywiście?

Pamiętajmy, że analiza nie polega na jak najwierniejszym odzwierciedleniu każdego detalu rzeczywistości, lecz na zamodelowaniu tych aspektów rzeczywistości, które są istotne dla naszego zastosowania. A w kontekście naszego systemu obsługi dopłat, zwierzę i działka, mimo że tak od siebie różne, mają jedną istotną cechę wspólną: są przedmiotami, za które przysługuje dopłata. I choćby z tego powodu warto stworzyć dla nich wspólną nadklasę, taką jak na Rysunku 4.

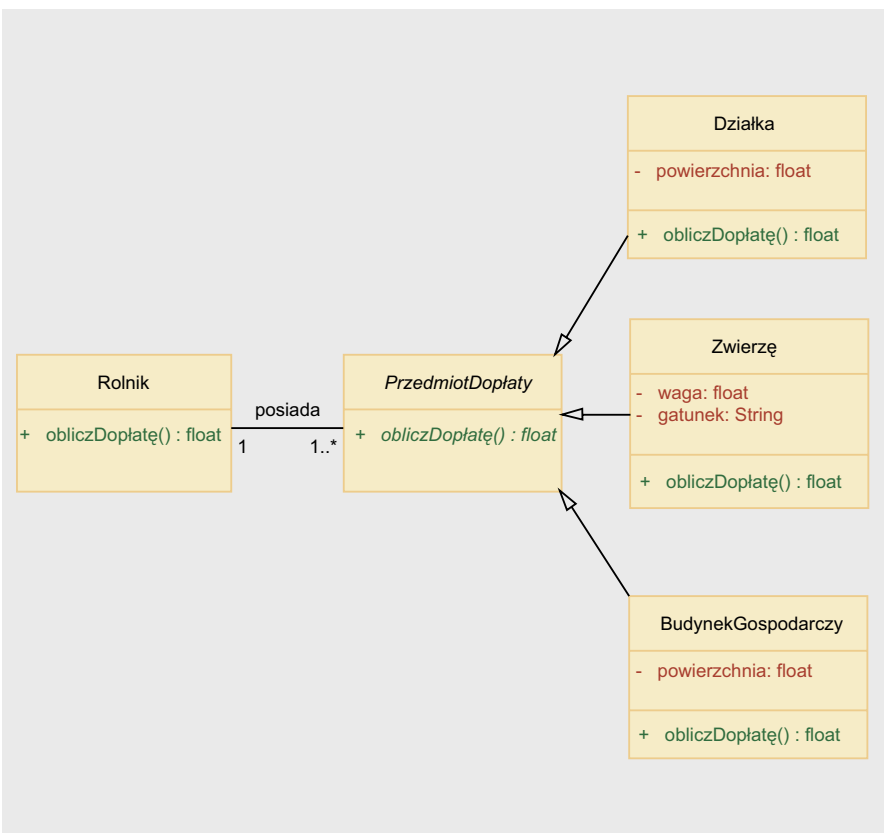
Zauważmy, że model ten jest w 100% zgodny z przyjętą definicją rolnika, który musi posiadać przynajmniej jeden przedmiot dopłaty. Ponieważ mamy teraz jedną relację posiadania zamiast dwóch, możemy spokojnie użyć liczebności `1..*`. Nie jest nam więc już potrzebna dodatkowa reguła biznesowa.

Model ten ma jeszcze jedną istotną zaletę: można go łatwo rozszerzyć o obsługę kolejnych przedmiotów dopłaty. Jeśli na przykład za jakiś czas zostaną wprowadzone dopłaty za budynki gospodarcze, wtedy wystarczy dodać kolejną podklasę klasy `PrzedmiotDopłaty`. Podklasa ta będzie posiadała swoją implementację metody `obliczDopłatę`, która wyliczy wysokość dopłaty np. w oparciu o powierzchnię użytkową budynku.

Taki rozszerzony model jest przedstawiony na Rysunku 5.

Zauważmy, że dodanie do naszego modelu nowej podklasy nie wymaga jakichkolwiek zmian w klasie `Rolnik`. Obiekt tej klasy, w celu obliczenia przysługującej dopłaty, po prostu sumuje dopłaty wyliczone dla wszystkich swoich obiektów posiadania, niezależnie od tego, czy są to działki, zwierzęta, budynki, czy jeszcze inne, dodane w przyszłości, obiekty.

W ten sposób, tworząc wspólną nadklasę dla dwóch klas pozornie zupełnie ze sobą niezwiązanych, udało nam się stworzyć model elegancki i rozszerzalny.



Rysunek 5. *Model systemu naliczania dopłat, rozszerzony o dodatkową podklasę*

Reguły Coad'a

Zasady dotyczące dziedziczenia, omówione w tym i w poprzednich odcinkach naszego cyklu, powinny pomóc Czytelnikom uniknąć typowych błędów związanych z dziedziczeniem. Stosując dziedziczenie warto też sprawdzić zasadność jego użycia przy pomocy tzw. reguł Coad'a. Jest to pięć podstawowych reguł sformułowanych przez Petera Coad'a, jednego z najbardziej znanych specjalistów zajmujących się UML-em i projektowaniem obiektowym.

Reguła 1. i częściowo 5. dotyczy problemu modelowania ról, przeanalizowanego w SDJ 5 i 6/2009. W SDJ 5/2009 omówiliśmy także regułę 2. Regułą 3. zajmiemy się w kolejnym odcinku naszego cyklu. Reguła 4. ma zaś charakter techniczny – dotyczy klas pomocniczych wykorzystywanych w implementacji (np. należących do bibliotek systemowych), więc na szczęście nie ma zastosowania przy tworzeniu modelu dziedziny biznesowej, w którym klasy pomocnicze zwyczajnie nie występują.

Reguły Coad'a

Dziedziczenia należy używać tylko wówczas, gdy spełnione są wszystkie poniższe warunki:

- Podklasa oznacza *bycie rodzajem* nadklasy, a nie *bycie rolą graną przez* nadklasę.
- Obiekt podklasy nigdy nie musi stać się obiektem innej klasy.
- Podklasa rozszerza odpowiedzialności nadklasy (a nie zmienia ich ani nie wyłącza).
- Podklasa nie rozszerza funkcjonalności klasy pomocniczej.
- W modelu dziedziny nadklasa reprezentuje rolę, transakcję lub urządzenie.

(Na podstawie: Bob Tarr, Some Object-Oriented Design Principles, *http://userpages.umbc.edu/~tarr/dp/lectures/OOPrinciples.pdf*)

W następnym odcinku

Za miesiąc poznamy polimorfizm – jeden z najważniejszych mechanizmów związanych z dziedziczeniem. Zobaczymy, jakie pułapki związane z polimorfizmem czyhają na nas podczas analizy.

SZYMON ZIOŁO

Główny konsultant w firmie RedPill, oferującej specjalistyczne szkolenia informatyczne oraz doradztwo w zakresie zarządzania projektami, modelowania procesów biznesowych oraz analizy i projektowania systemów informatycznych. Analityk biznesowy i systemowy, specjalizujący się w analizie obiektowej, modelowaniu w języku UML oraz w XML-u i technologiach pokrewnych.

Kontakt z autorem: sziolo@redpill.com.pl

W sprzedaży w salonach prasowych
w całej Polsce

więcej informacji na www.sdjournal.org

Prawo autorskie dla programistów

Programista jako twórca dzieła w postaci programu komputerowego

W przypadku, gdy programista tworzy program komputerowy, który jest przejawem jego działalności twórczej, ma indywidualny charakter i jest rezultatem jego pracy, wówczas taki program należy traktować jako utwór w rozumieniu Prawa autorskiego. Sam programista powinien być natomiast traktowany jako twórca.

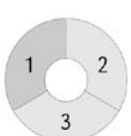
Dowiesz się:

- Jakie cechy powinien posiadać program, aby stanowić przedmiot prawa autorskiego;
- Jakie znaczenie ma fakt wspólnego tworzenia programu komputerowego;
- Z czym wiąże się tworzenie programu komputerowego w ramach stosunku pracy.

Powinieneś wiedzieć:

- Twórcą może być tylko osoba fizyczna, natomiast majątkowe prawa autorskie mogą przysługiwać również osobie prawnej;
- Zapisy umowy o pracę z programistą nie muszą być dla niego niekorzystne.

Poziom trudności



Przepisy ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tj. Dz. U. z 2006 r., Nr. 90, poz. 631 ze zm.), zwanej dalej Prawem autorskim, nie wprowadzają wprost definicji *twórcy*, a tym bardziej definicji *programisty*. Wydaje się, iż z samej natury procesu twórczego wynika fakt, iż twórcą może być wyłącznie osoba fizyczna. W kontekście jednakże specyfiki tworzenia programów komputerowych mogą jednak zaistnieć na tym tle wątpliwości, związane z tym, iż nie da się ukryć, iż bez pomocy maszyny, jaką jest komputer, powstanie tego typu dzieł nie byłoby ani możliwe, ani też celowe. Niemniej jednak, aktualny stan prawny nie pozwala uznać maszyny jako twórcy określonego dzieła (w tym programu komputerowego). Do czasu powstania tzw. sztucznej inteligencji, mogącej w pełni zastąpić człowieka w zakresie procesu twórczego (do którego zaliczyć należy też sam pomysł, koncepcję, cele) i prawnego uregulowania statusu takich *przedmioto-podmiotów*, rozważanie możliwości zakwalifikowania maszyny jako twórcy czy też wyeliminowania programisty jako autora programu komputerowego, stanowi jedynie *prawnicze science-fiction*.

Brak ustawowej definicji twórcy wynika z faktu, iż to, czy dane dzieło ma cechy utworu, wynika nie z cech jego twórcy, ale z cech samego dzieła. Tworzenie przez ustawodawcę definicji twórcy mogłoby stanowić o odwróceniu tej słusznej zasady. Zgodnie bowiem z art. 1 ust. 1 Prawa autorskiego utworem jest każdy przejaw działalności twórczej o indywidualnym charakterze, ustalony w jakiejkolwiek postaci, niezależnie od wartości, przeznaczenia i sposobu wyrażenia. Zgodnie z art. 1 ust. 2, w szczególności przedmiotem prawa autorskiego są utwory wyrażone słowem, symbolami matematycznymi, znakami graficznymi (literackie, publicystyczne, naukowe, kartograficzne oraz programy komputerowe).

Twórcą może być więc nazwany tylko ten programista, który jest autorem programu komputerowego:

- stanowiącego rezultat jego pracy;
- stanowiącego przejaw działalności twórczej (rezultat działalności o charakterze kreacyjnym);
- mającego indywidualny charakter (oryginalny, niepowtarzalny).

Domniemywa się, że twórcą jest osoba, której nazwisko w tym charakterze uwidoczniono na egzemplarzach utworu lub której autorstwo podano do publicznej wiadomości w jakikolwiek inny sposób w związku z rozpowszechnianiem utworu. Celem takiego domniemania jest jedynie ułatwienie ewentualnego ustalenia osoby twórcy. Jak każde domniemanie prawne może być więc ono obalone. Dla takiego obalenia wystarczy udowodnienie, że osoba, której nazwisko figuruje na egzemplarzu utworu (np. na nośniku oprogramowania lub w informacji zawartej w ramach interfejsu programu), nie wniosła twórczego wkładu w tworzenie programu lub że wkład taki wniosła również inna osoba (w przypadku braku oznaczenia faktu współautorstwa programu).

Zgodnie z art. 8 ust. 1 ww. ustawy, prawo autorskie przysługuje twórcy, o ile ustawa nie stanowi inaczej. Zasada ta dotyczy zarówno praw majątkowych, jak i osobistych. Wyjątki od niej mogą wynikać wyłącznie z ustawy, a nie z woli stron. Jeden z takich wyjątków został uregulowany w rozdziale 2 Prawa autorskiego i dotyczy utworu zbiorowego, do którego autorskie prawa majątkowe przysługują producentowi lub wydawcy (art. 11 Prawa autorskiego). Drugi wyjątek, dotyczący właśnie autorskich praw majątkowych do programu komputerowego, zostanie omówiony w dalszej części niniejszego artykułu.

Poza sytuacjami określonymi w Prawie autorskim, inna osoba może stać się podmiotem prawa autorskiego poprzez nabycie tego prawa od twórcy w drodze umowy o przeniesienie majątkowych praw autorskich albo w drodze dziedzi-

czenia. Takie nabycie prawa nie ma jednak charakteru pierwotnego, nie dotyczy bowiem powstania praw autorskich, tylko wynika z ich przeniesienia.

Współautorstwo programu

Specyfika programów komputerowych powoduje, że różne ich części (aplikacje, moduły itd.) mogą być tworzone, a czasem nawet wymagają współpracy wielu osób. Zgodnie z art. 9 ust. 1 Prawa autorskiego, współtwórcom przysługuje prawo autorskie wspólnie. Domniemywa się, że wielkości udziałów (w tym prawie) są równe. Każdy ze współtwórców może żądać określenia wielkości udziałów przez sąd, na podstawie wkładów pracy twórczej. Ponadto, każdy ze współtwórców może wykonywać prawo autorskie do swojej części utworu mającej samodzielne znaczenie, bez uszczerbku dla praw pozostałych współtwórców (w szczególności może mieć to znaczenie w przypadku tzw. *programów wielomodułowych*). Do wykonywania prawa autorskiego do całości utworu potrzebna jest zgoda wszystkich współtwórców. W przypadku braku takiej zgody każdy ze współtwórców może żądać rozstrzygnięcia przez sąd, który orzeka, uwzględniając interesy wszystkich współtwórców.

Natomiast każdy ze współtwórców może dochodzić roszczeń z tytułu naruszenia prawa autorskiego do całości utworu. Uzyskane w drodze dochodzenia tych roszczeń świadczenie przypada wszystkim współtwórcom, stosownie do wielkości ich udziałów.

Zgodnie z art. 9 ust. 5 Prawa autorskiego, do autorskich praw majątkowych przysługujących współtwórcom stosuje się odpowiednio przepisy Kodeksu cywilnego o współwłasności w częściach ułamkowych. Takimi przepisami będą na przykład przepisy o zniesieniu współwłasności. W odniesieniu do programów komputerowych zastosowanie może mieć zwłaszcza art. 212 ust. Kodeksu cywilnego, zgodnie z którym rzecz, która nie daje się podzielić, może być przyznana stosownie do okoliczności jednemu ze współwłaścicieli z obowiązkiem spłaty pozostałych albo sprzedana stosownie do przepisów Kodeksu postępowania cywilnego.

Należy zauważyć, iż ustalenie kręgu współautorów programu komputerowego nie może być wynikiem ustaleń umownych (nie ogranicza to jednak możliwości przeniesienia udziału w majątkowych prawach do programu). Jedynie stan faktyczny może być podstawą dla wiarygodnego ustalenia kręgu współtwórców programu. W szczególności ustalenie takie powinno obejmować zakres i rodzaj dokonanej przez potencjalnego współautora pracy o charakterze twórczym.

W odniesieniu do programu współtworzonego przez wiele osób, z mocy ustawy, powstaje więc na rzecz jego programistów w sposób pierwotny wspólne autorskie prawo majątkowe. W kontekście uwag poczynionych w dalszej części artykułu nie można też wykluczyć sytuacji, iż wspólne prawo majątkowe powstać może także na rzecz różnych pracodawców zatrudniających współautorów danego programu komputerowego, jak również jednocześnie w odniesieniu do jednego z współtwórców oraz osoby będącej pracodawcą drugiego.

Należy zauważyć, iż w zakresie współautorstwa istnieje wiele wątpliwości natury prawnej. W szczególności zarówno doktryna prawa, jak i orzecznictwo sądów nie dopracowały się jednolitej koncepcji, czy aby doszło do współtworzenia dzieła, konieczne jest zawarcie pomiędzy twórcami porozumienia w tym zakresie. W mojej ocenie takie porozumienie nie jest wymagane, jednakże za współautora nie można uznać danej osoby wbrew jej woli.

O ile powyższy problem wydaje się być głównie teoretycznym, o tyle praktycznym (zwłaszcza w odniesieniu do programów komputerowych) może być problem zakwalifikowania osoby tworzącej poprawki do programu lub wprowadzającej do niego zmiany. O ile osoba poprawiająca jedynie błędy w programie raczej nie może, w mojej ocenie, pretendować do roli jego współtwórcy, o tyle postawienie takiej tezy w stosunku do osoby, która jest uprawniona do dokonania zmiany w programie komputerowym, może być uzasadnione okolicznościami faktycznymi. Zakres zmian może bowiem spowodować, iż powstanie właściwie nowy utwór. W takim przypadku trudno odmówić wprowadzającemu zmiany w programie cech pozwalających na uznanie go jako współautora.

Pracodawca jako podmiot majątkowych praw autorskich

Zgodnie z art. 74 ust. 3 Prawa autorskiego, prawa majątkowe do programu komputerowego stworzonego przez pracownika w wyniku wykonywania obowiązków ze stosunku pracy przysługują pracodawcy, o ile umowa nie stanowi inaczej. Przepis ten wprowadza zasadę pierwotnego nabycia majątkowych praw autorskich przez pracodawcę twórcy w tych przypadkach, gdy program został stworzony przez pracownika przy wykonywaniu obowiązków ze stosunku pracy.

Należy zauważyć, iż przepis ten ma zastosowanie wyłącznie w przypadku istnienia stosunku pracowniczego pomiędzy twórcą programu a innym podmiotem. Zgodnie z ogólnie obowiązującą zasadą prawa, iż wyjątków nie należy interpretować rozszerzająco, regulacja ta nie będzie miała odpowiedniego zastosowania do umów o charakterze zbliżonym do umowy o pracę (np. umowa o dzieło, umowa zlecenia).

Jednakże, zgodnie z wyrokiem Sądu Najwyższego z dnia 9 stycznia 2001 r. (sygn. akt I PKN 493/00), *właścicielem praw autorskich do programu komputerowego wykonanego w ramach obowiązków pracowniczych jest pracodawca, chyba że w umowie o pracę lub umowie cywilnoprawnej strony postanowiły inaczej*. Taka teza może jednak mieć uzasadnienie w przypadku, gdy oprócz umowy o pracę, twórcę programu i jego pracodawcę łączyłby jeszcze inny stosunek prawny poza pracowniczym.

Niezależnie od powyższego, należy stwierdzić, iż jedynie odpowiednie zapisy umowy o pracę lub innej umowy zawartej niejako „równolegle" mogłyby wykluczyć nabycie przez pracodawcę majątkowych praw autorskich do programu utworzonego przez programistę w ramach jego obowiązków pracowniczych. Nie ma więc przeszkód, aby na zasadzie swobody umów strony umowy ustaliły, iż część, a nawet całość majątkowych praw autorskich do programu przysługuje twórcy, a nie pracodawcy. Jeżeli istnieje więc taka potrzeba, programista powinien dopilnować, aby tak istotne zapisy znalazły się w podpisywanej przez niego umowie. W praktyce jednak zgoda pracodawcy na takie odstępstwo związana być może z zakresem prowadzonej przez niego działalności. Wątpliwym więc wydaje się, aby zgodził się on na takie ukształtowanie wzajemnych praw i obowiązków np. w sytuacji, gdy dla celów prowadzonej przez niego działalności elementarne znaczenie ma dysponowanie pełnymi prawami majątkowymi do programów tworzonych przez jego pracowników. Nie można jednak wykluczyć takiej sytuacji, gdy pracodawca skłonny będzie do odstąpienia od przysługujących mu ustawowo uprawnień przynajmniej w zakresie części majątkowych praw autorskich. Brak jest jednak przepisów, które wprowadzałyby po stronie pracodawcy obowiązek rezygnacji z tych uprawnień.

Warto również zaznaczyć, iż art. 14 wprowadza odmienne zasady nabywania majątkowych praw autorskich i korzystania z nich niż art. 74 ust. 3 Prawa autorskiego. Należy jednak stwierdzić, iż w sytuacji, gdy program komputerowy stanowi jednocześnie *utwór dydaktyczny* regulacje szczególne (art. 74 ust. 3) dotyczące programów komputerowych wyłączają zastosowanie art. 14 Prawa autorskiego. Nie ma jednak przeszkód, aby również w takim przypadku umowa o pracę zawierała postanowienia bardziej korzystne dla programisty – pracownika naukowego, w tym również analogiczne do regulacji zawartych w art. 14 ustawy.

PIOTR PRZEGALIŃSKI

Radca prawny, absolwent Wydziału Prawa, Prawa Kanonicznego i Administracji Katolickiego Uniwersytetu Lubelskiego. Prowadzi indywidualną Kancelarię Radcy Prawnego w Warszawie. Posiada doświadczenie w obsłudze prawnej przedsiębiorców, w tym również oferujących zaawansowane narzędzia i usługi informatyczne dla przedsiębiorstw.

Kontakt z autorem: piotrprzegalinski@krpp.pl, www.krpp.pl

KLUB PRO

Oferta skierowana dla firm

Jeżeli Twoja firma jest prenumeratorem Software Developer's Journal za comiesięczną dopłatą 50 PLN +VAT możesz dodatkowo otrzymać reklamę.
Wystarczy tylko, aby profil Twojej firmy pokrywał się z naszym magazynem.

Wyślij do nas: logo firmy, dane kontaktowe i informację o firmie

Reklama przez 12 kolejnych numerów tylko za 600 PLN +VAT.

Jeżeli nie posiadasz jeszcze prenumeraty możesz ją zamówić w atrakcyjnej cenie.

Dla nowych prenumeratorów specjalna oferta – 690 PLN.

Skontaktuj się z nami:
tel. +48 22 877 20 80
fax: +48 22 877 20 70
software@europress.pl

Opera Software

Opera Software's vision is to deliver the best Internet experience on any device. We are offering browser for PC/desktops and embedded products that operates across devices, platforms and operating systems. Our browser can deliver a faster, more stable and flexible Internet experience than its competitors.

http://www.opera.com

Architektury systemów IT

Twórca frameworków JUVE i serwera aplikacji AVAX oferuje usługi, doradztwo, rozwiązania do tworzenia nowoczesnych, dużych systemów i rozwiązań informatycznych/internetowych, integrujące architektury ery post-J2EE/.NET, wykorzystujące MDD/MDA dla dziedzin – bankowość, telekomunikacja, handel, e-commerce, ERP/Workflow/CRM, rozwiązania internetowe, portalowe.
www.mpsystem.com mpsystem@mpsystem.com

FRONTLINE STUDIOS

Frontline Studios – amerykańsko-polski twórca gier konsolowych oraz PC – szuka utalentowanych koderów (bardzo dobra znajomość C/C++; Java, Delphi, UML), doświadczonych grafików (2D, 3D) oraz zespołów developerskich. Oferujemy długoterminową pracę nad poważnymi projektami. Nie przegap tej okazji!

kadry@frontstudios.com

Future Processing

Future Processing to dynamiczna firma technologiczna działająca na globalnym rynku oprogramowania. Jesteśmy zespołem wysokiej klasy specjalistów posiadających wiedzę i doświadczenie niezbędne do realizacji ambitnych projektów informatycznych. Jeśli programowanie to Twoja pasja dołącz do nas! (możliwość pracy zdalnej).

http://www.future-processing.pl

Kei.pl

Kei.pl działa na rynku usług hostingowych od 2000 roku. Do naszych zadowolonych Klientów z dumą możemy zaliczyć wiele przedsiębiorstw sektora MSP, instytucji oraz osób prywatnych. W ofercie Kei.pl znajdują się pakiety hostingowe, a także usługi dla wymagających Użytkowników – platformy *e-Biznes oraz serwery fizyczne.*

http://www.kei.pl

WSISiZ w Warszawie

INFORMATYKA ZARZĄDZANIE
studia stopnia I i II (stacjonarne i niestacjonarne) specjalności: inżynierskie, magisterskie i licencjackie. Szczegółowe plany studiów, opisy poszczególnych specjalności – zapraszamy na stronę uczelni.

http://www.wit.edu.pl

Playsoft

Playsoft jako lider portowania aplikacji na platformy mobilne wciąż powiększa bazę swoich klientów: EA Mobile, Sega, THQ, Konami. W ramach rozszerzania swojej działalności, poszukujemy doświadczonego programisty, który byłby odpowiedzialny za tworzenie aplikacji na platformy Iphone, Windows Mobile, Android.
http://www.playsoft.fr

KLUB PRO

INFOTEX SP.J Śmietanowski i Wsp.

Dystrybutor XP Unlimited – Serwer Terminali dla Windows XP i VISTA. Program umożliwia łączenie się z dowolnego klienta Windows, Linux z wykorzystaniem protokołu RDP. Cena wersji Classic dla 5 użytkowników - 165€, dla nieograniczonej liczby - 235€. Ponadto oferujemy opiekę serwisową i aplikacje internetowe na zamówienie.

http://www.infotex.com.pl

StatConsulting

StatConsulting to firma o znaczącej pozycji na rynku usług analitycznych oraz Data Mining. Nasza oferta obejmuje m.in. modele scoringowe, rozwiązania Fraud Detection oraz modelowanie ryzyka. Tworzymy także własne rozwiązania informatyczne na potrzeby projektów Data Mining, Data Quality, zarządzania ryzykiem itd.

http://www.statconsulting.com.pl

TTS Company Sp. z o.o.

Sprzedaż i dystrybucja oprogramowania komputerowego. Import programów na zamówienie. Ponad 200 producentów w standardowej ofercie. Chcesz kupić oprogramowanie i nie możesz znaleźć polskiego dostawcy? Skontaktuj się z nami – sprowadzimy nawet pojedyncze licencje.

http://www.OprogramowanieKomputerowe.pl

IT SOLUTIONS

Wdrożenia i szkolenia z zakresu:

- SQL Server
- SharePoint Services
- MS Project / Server
- Tworzenie aplikacji w technologii .NET

http://www.itsolutions.biz.pl
marcin.pytlik@itsolutions.biz.pl

Softline rozwiązania mobilne

Wiodący producent systemów mobilnych, dostawca aplikacji użytkowych dla biznesu (Symbian OS, Windows Mobile, J2ME) zaprasza do współpracy. Zostań naszym partnerem. Dołącz do zespołu.

http://www.softline.com.pl

Proximetry Poland Sp. z o.o.

Proximetry Poland Sp. z o.o. jest polskim oddziałem amerykańskiej firmy Proximetry Inc. – dostawcy systemów zarządzania sieciami bezprzewodowymi opartymi na technologiach WiFi i WiMAX. Naszą misją jest dostarczenie klientom rozwiązań poprawiających jakość usług (QoS) dostarczanych drogą radiową. Dołącz do najlepszych i zostań członkiem naszej ekipy!

http://www.proximetry.com

SWD Software Sp. z o.o

System operacyjny czasu rzeczywistego RTOS QNX. Oficjalny dystybutor w Polsce.
Zakres działalności: lokalizacja produktów QNX, dostawy sprzętu i oprogramowania, doradztwo przedsprzedażowe, wsparcie techniczne, certyfikowane szkolenia, opracowania na zamówienie.

http://www.swdsoft.pl

Systemy bankowe, ISOF

HEUTHES istnieje na rynku od 1989 r. Obok systemów informatycznych dla banków, oferuje nowoczesne oprogramowanie do obsługi firm. System ISOF jest udostępniany klientom w trybie SaaS lub licencji. Pracuje na platformie Linux i zawiera m.in. takie moduły jak CRM, DMS, Magazyn, Sprzedaż, Logistyka oraz Rachunkowość.

http://www.isof.pl

Roczna prenumerata

tylko 250,-

Software Developer's Journal (poprzednio Software 2.0) jest miesięcznikiem głównie dla programistów, którzy liczą, że dostarczymy im gotowe rozwiązania, oszczędzając im czasu i pracy. Jesteśmy czytani przez tych, którzy chcą być na bieżąco informowani o najnowszych osiągnięciach w dziedzinie IT i nie chcą, żeby jakiekolwiek istotne wydarzenia umknęły ich uwadze. Aby zadowolić naszych czytelników, prezentujemy zarówno najnowsze rozwiązania, jaki starsze, sprawdzone technologie.

UWAGA!

Zmiana danych kontaktowych

Obsługa prenumeraty

1. Telefon
+48 22 877 20 80
2. Fax
+48 22 877 20 70
2. Online
software@europress.pl

3. Adres
EuroPress Polska Sp. z o.o.
ul. Jana Kazimierza 46/54
01-248 Warszawa

Zamówienie prenumeraty

Zadzwoń
+48 22 877 20 80
lub
zamów
mailowo!

Prosimy wypełniać czytelnie i przesyłać faksem na numer:
00 48 22 877 20 70
lub listownie na adres:
EuroPress Polska Sp. z o.o.
ul. Jana Kazimierza 46/54
01-248 Warszawa
Polska
E-Mail: *software@europress.pl*

Przyjmujemy też zamówienia telefoniczne:
00 48 22 877 20 80

Jeżeli chcesz się dowiedzieć o formach płatności, wejdź na stronę:
www.europress.pl lub napisz na e-mail: *software@europress.pl*

Imię i nazwisko .....

Nazwa firmy.....

Dokładny adres .....

.....

Telefon .....

E–mail .....

ID kontrahenta .....

Numer NIP firmy .....

Fax (wraz z nr kierunkowym) .....

□ automatyczne przedłużenie prenumeraty

Prenumerujesz – zyskujesz

- oszczędność pieniędzy
- szybka dostawa
- prezenty
- bezpieczna płatność on–line

Tytuł	Ilość numerów	Ilość zamawianych prenumerat	Od numeru pisma lub miesiąca	Cena
Software Developer's Journal (1 płyta CD) – dawniej Software 2.0	**12**			**250 PLN**

Software Developer's JOURNAL
new ideas & solutions for professional programmers

W NASTĘPNYM NUMERZE SOFTWARE DEVELOPER'S JOURNAL 9/2009 (177)

Kolejny numer SDJ poświęcimy Bazom danych

W numerze znajdą się między innymi artykuły:

- SQL Server Data Access Components
- Bazy danych – współpraca z popularnymi technologiami

Do współpracy zaprosiliśmy również firmy IBM I Oracle.

Ponadto w magazynie:

- Klub Techniczny Adobe
- Stałe elementy:
  - Prawo w IT,
  - Akademia UML,
  - Porady dla programistów.

W sprzedaży w drugiej połowie sierpnia

Nowa seria Video Kursów na CD poświęconych Javie

Redakcja zastrzega sobie prawo do zmiany zawartości magazynu

Software Developer's JOURNAL
new ideas & solutions for professional programmers

Adobe

BOGATE MOŻLIWOŚCI OTWARTYCH ROZWIĄZAŃ

ODKRYJ ADOBE FLEX > www.adobe.com/go/flex

Program ADOBE ® FLEX® 3 to wysokowydajne środowisko typu Open Source do tworzenia i obsługi wyrazistych aplikacji internetowych, które w sposób jednolity można wdrażać dla większości najważniejszych przeglądarek , komputerów oraz systemów operacyjnych. Chociaż aplikacje Flex można tworzyć za pomocą darmowego pakietu Flex SDK, programiści mogą korzystać z oprogramowania Flex Builder ™ 3, które istotnie przyspiesza proces opracowywania aplikacji.

< NIE ZWLEKAJ: PRZYSPIESZ SWOJĄ PRACĘ DZIĘKI ADOBE FLEX BUILDER 3 >-

ON.

Software Developer's JOURNAL
new ideas & solutions for professional programmers

BIBLIA PROGRAMISTY

PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

220 STRON NOWYCH DOGMATÓW

WIĘCEJ INFORMACJI: LUKASZ.LOPUSZANSKI@SOFTWARE.COM.PL