XVI Konferencja PLOUG
Kościelisko
Październik 2010

Czy świat jest obiektowy?

Tomasz Wyrozumski
BMS Creative

tw@bms.krakow.pl

Abstrakt. Artykuł dotyczy podstawowych zagadnień analizy i projektowania systemów. Zastanawiamy się w nim nad genezą tzw. Myślenia obiektowego i nie negując zalet obiektowości, polemizujemy z niektórymi rozpowszechnionymi poglądami na jej temat. Argumentujemy, że nadużywanie takiego podejścia oddala analityków i projektantów od rozumienia rzeczywistości, której dotyczą budowane systemy, a tym samym niekorzystnie wpływa na ich kształt. Jako alternatywę proponujemy oparte na ekstrakcji reguł koncepcje ontologiczne, które – jak wierzymy – doczekają się kiedyś skutecznej implementacji w sferze technologii.

Informacja o autorze. Z wykształcenia fizyk, absolwent Uniwersytetu Jagiellońskiego. Doktorat w dziedzinie fizyki teoretycznej na Uniwersytecie Wiedeńskim. Od 1992 roku zajmuje się profesjonalnie informatyką, będąc współwłaścicielem i dyrektorem BMS Creative. Aktywnie kieruje projektami komercyjnymi w zakresie systemów CRM, ERP, obiegu dokumentów, zarządzania infrastrukturą teleinformatyczną oraz eksploracji danych, w szczególności z wykorzystaniem sztucznych sieci neuronowych.

1. Rozmowy kwalifikacyjne

Za genezę tego artykułu należałoby uznać rozmowy kwalifikacyjne, które autorowi zdarzało się przeprowadzać z osobami pragnącymi zdobyć pracę w charakterze programisty. Sam będąc programistą od dawna niepraktykującym nie zadawał im szczegółowych pytań z zakresu technologii (to stanowiło domenę młodszych kolegów), lecz poruszał tzw. „kwestie ogólnorozwojowe". Chodziło więc w tym wypadku nie o to, by stwierdzić, czy kandydat np. wie jak użyć Webservices, ile raczej o to, by ustalić, jaka jest jego motywacja do podjęcia takiej, a nie innej pracy, jak komunikuje się z otoczeniem, jak szerokie ma horyzonty itp. Jedno z pytań dotyczyło programowania obiektowego i brzmiało (przynajmniej na pozór) bardzo naiwnie: po co je wymyślono? I tu niezwykle często padały odpowiedzi, które dla autora brzmiały nader zaskakująco, a sprowadzały się mniej więcej do stwierdzenia, iż programowanie obiektowe stanowi naturalną pochodną analizy i projektowania obiektowego, a nawet szerzej, myślenia obiektowego, które idealnie nadaje się do modelowania rzeczywistości. Dlaczego? No, bo w istocie rzeczywistość po prostu jest obiektowa…

Ten fascynujący pogląd wydaje się być wśród młodych ludzi na tyle powszechny, że wytłumaczenie, iż ktoś na jakiejś uczelni tak właśnie wykłada, należy uznać za całkowicie niewystarczające. Wykładać tak musi wielu ludzi na wielu uczelniach, a to już powód, by rozpocząć na ten temat poważną dyskusję.

Przeglądając wstępy (kto je czyta?) do kilku popularnych pozycji z dziedziny technik obiektowych ([CoYo94], [DKK98], [Jasz97], [MaOd97], [Tayl94]) zauważamy w nich pewne nieśmiałe wzmianki na temat obiektowości otaczającego nas świata, choć, uczciwie mówiąc, w żadnej ze wspomnianych książek nie napisano tego, co słyszy się od młodych informatyków. Sięgnijmy zatem do *Wikipedii*, która – z całym szacunkiem dla jej użyteczności oraz zawartej tam ogromnej ilości niekwestionowanych faktów – stanowi też swoisty fenomen socjologiczny, i pozwala wywnioskować coś na temat panujących w różnych środowiskach poglądów. I tak, możemy przeczytać w niej, co następuje:

Największym atutem programowania, projektowania oraz analizy obiektowej jest zgodność takiego podejścia z rzeczywistością – mózg ludzki jest w naturalny sposób najlepiej przystosowany do takiego podejścia przy przetwarzaniu informacji. Już Platon postulował dwoistość: „byt" – „idea (wzorzec) bytu", np. książka jako idea ogólna – w terminologii platońskiej: doskonała (choć niedookreślona), i książka konkretna np. zbiór baśni leżący na piątej półce. Podobnie, choć później Arystoteles analizując rzeczywistość wprowadził pojęcia formy i materii.

Ten krótki fragment tekstu zawiera tyle kategorycznych stwierdzeń z dziedziny informatyki, psychologii, neurofizjologii oraz filozofii, że w każdym czytelniku wzbudzić musi podziw dla głębokiej i wszechstronnej wiedzy jego autora lub raczej autorów. Naszym celem nie jest oczywiście polemika z taką, czy inną opinią jakiegoś filozofa amatora, bo na to szkoda czasu, lecz raczej krytyczne przedyskutowanie poglądów rozpowszechnionych w większej zbiorowości, a przez to wywierających wpływ na rozwój określonej dziedziny ludzkiej aktywności – w tym przypadku informatyki. Nie będziemy więc specjalnie „czepiać się" szczegółów i zostawimy w spokoju biednego Platona, czy Arystotelesa, natomiast spróbujemy zastanowić się nad głębszym sensem owej – jak się wydaje – reprezentatywnej dla pewnego środowiska wypowiedzi. Jak duże jest to środowisko? Tego oczywiście dokładnie nie wiemy, choć dysponujemy pewną wskazówką. Otóż nie tylko szacowna *Encyclopaedia Britannica* w ramach hasła *Object Oriented Programming* nie oferuje swoim czytelnikom podobnych bzdur, ale nie propaguje ich również *Wikipedia* anglojęzyczna. Autor zadał sobie nawet nieco trudu, by sprawdzić wersję niemiecką, francuską, włoską i rosyjską i w żadnej z nich nie znalazł równie absurdalnych stwierdzeń, jak w wersji polskiej. Z jednej strony jest to pocieszające, że stopień rozpowszechnienia nonsensów zdaje się być ogra-

niczony, z drugiej jednak – dlaczego właśnie do naszego obszaru językowego?! Tym bardziej więc zasługuje na rozważenie pytanie zawarte w tytule naszego artykułu, napisanego, jakby nie było, w języku polskim.

Czy zatem świat jest obiektowy? Albo może, czy obiektowy charakter ma nasze poznanie? W praktyce wychodzi na jedno, ponieważ już od czasów Immanuela Kanta wiemy, że rzeczywistość stanowi niewiadomą (*Ignotum X*), zaś wszelkie nasze stwierdzenia na jej temat odnoszą się do obrazu owej rzeczywistości, wytworzonego w naszym mózgu, który to obraz może mieć się nijak do pierwowzoru. Ta kwestia nie ma jednak akurat nic wspólnego z obiektowością świata, lecz stanowi wspólny metodologiczny problem całego przyrodoznawstwa.

Ponawiamy więc pytanie w owej uproszczonej formie*: Czy świat jest obiektowy?* i w dalszej części artykułu próbujemy na nie odpowiedzieć.

2. Krótka historia programowania

Pierwsze elektroniczne maszyny liczące powstały tak niedawno, że wciąż żyją wśród nas (i świetnie się trzymają) ludzie pamiętający dni, gdy maszyn tych jeszcze nie było na świecie. Z drugiej strony, dla najmłodszego pokolenia czasy bez Internetu wydają się co najmniej tak samo odległe, jak epoka wojen napoleońskich. Mamy więc nadzieję, że nie będzie wielkim faux pas przypomnieć tutaj początki informatyki.

I tak, programowanie pierwszych maszyn cyfrowych sprowadzało się do wydawania im bardzo dokładnych, „technicznych", by tak rzec, instrukcji, co mają robić. Przykładowo, chcąc by komputer zsumował dwie liczby, programista musiał nakazać mu pobranie ich z odpowiednich komórek pamięci, wprowadzenie do właściwych rejestrów procesora, wykonanie operacji dodawania oraz przeniesienie wyniku pod wskazany adres. Choć niektórym trudno dziś sobie to wyobrazić, mniej więcej tak to się wówczas odbywało, zaś ASSEMBLER stanowił już pewną formę abstrakcji dla owych operacji. Formą znacznie wyższą stał się język FORTRAN, choć każdy, kto miał przyjemność się z nim zetknąć, zauważy pewne swoiste podobieństwo do ASSEMBLERA (instrukcje skoku).

Wyraźnym odejściem od pierwowzoru stał się dopiero paradygmat strukturalny oraz języki takie jak np. PASCAL, ALGOL, czy C. Skakanie uznano w nich za szczyt nieelegancji, a zamiast niego zaproponowano tworzenie przejrzystych konstrukcji logicznych. Gdyby ludzie mówiąc po angielsku zaczęli porozumiewać się między sobą używając zdań formalnych i odrzucając wszelkie niejednoznaczności w swoich wypowiedziach, to prawdopodobnie przeszliby np. na PASCAL, jako najbliższy językowi naturalnemu. Osiągnęliśmy niewątpliwie wielki postęp od czasu, gdy konieczne było posługiwanie się rejestrami i adresami pamięci. Ale co dalej?

W tym miejscu informatyka stanęła jakby na rozdrożu. Było oczywiste, że języki programowania powinny rozwijać się tak, by ułatwiać pracę programistom. Ale co to oznacza? Wydaje się, że powinny one niejako odrywać programistów od maszyn, zaś kierować ich uwagę w stronę rozwiązywanego problemu. Nie powinni więc oni interesować się tranzystorami, układami scalonymi, rejestrami procesora, adresowaniem pamięci itd., lecz raczej rzeczywistością odwzorowywaną w programy. Tak powstały języki deklaratywne, w rodzaju SQL. Mówi się, że w językach tych programista nie opisuje sekwencji działań prowadzących do osiągnięcia pewnego rezultatu, lecz sam rezultat, chociaż przy dokładniejszej analizie stwierdzenie to wydaje się raczej płytkie i mało precyzyjne. W istocie od języków strukturalnych odróżnia je stopień specjalizacji, czyli bardzo ograniczony zakres stosowalności. Zdanie w rodzaju SELECT … FROM … WHERE nadal jest poleceniem wydanym maszynie, podobnie jak IF … THEN … ELSE. Nie tyle więc opisujemy rezultat, co raczej wydajemy bardziej wyspecjalizowane rozkazy, ponieważ korzystamy z narzędzia, jakim jest system zarządzania bazą danych.

Nawiasem mówiąc, język SQL wymyślono nie dla programistów, ale dla użytkowników! Na starych filmach widzi się niekiedy szpiegów, którzy zakradają się do terminali i z niezwykłą zręcznością wystukują na klawiaturze jakieś formuły, po których drukarka wypluwa np. listę wszystkich agentów CIA działających na Bliskim Wschodzie. Owi szpiedzy używają właśnie języka SQL! Dziś rozpaskudzeni użytkownicy „wyklikują" analogiczne formuły korzystając z okienek, jednak w rzeczy samej niewiele się tu zmieniło od czasu, gdy wymyślono paradygmat strukturalny. Używamy coraz bardziej wyspecjalizowanych narzędzi, ale nadal musimy dostosować się do ich specyfiki, a więc wyrażać się ściśle i precyzyjnie, konstruując poprawne zdania logiczne.

Drugą opcją byłoby dostosowanie się maszyn do nas, a więc spowodowanie, by zaczęły one rozumieć język naturalny. Nie rozwijamy tego wątku, gdyż mógłby on z powodzeniem stanowić temat odrębnego, obszernego artykułu. Pozostaniemy tylko przy stwierdzeniu, że mimo znacznych nakładów pracy niezbyt wiele udało się w tej materii osiągnąć.

Pojawił się natomiast trzeci, ważny nurt, związany ze wspomnianym już we wstępie podejściem obiektowym. Jego początki sięgają jeszcze lat sześćdziesiątych, natomiast za pewien przełom uznać można rok 1970, kiedy to pracujący dla firmy Xerox Alan Kay wraz ze swoim zespołem stworzył pierwszy w pełni obiektowy język programowania, SMALLTALK. Tym właśnie trzecim nurtem zamierzamy się teraz zająć, przypominając na początek kilka oczywistych faktów.

3. Czym naprawdę jest obiektowość?

Istotą paradygmatu obiektowego jest bardzo ścisłe powiązanie danych z procedurami służącymi do ich przetwarzania. Chodzi więc o to, by dane niejako wiedziały, jak się przetwarzać lub, mówiąc bardziej konkretnie, by programista z rozpędu nie potraktował określonych danych inną, niżby należało procedurą. Obiekty składają się więc z pól i metod, zaś każdy z nich stanowi zamkniętą całość, do której wnętrza nie można się dostać jakimiś bocznymi drzwiami. Zjawisko to, zwane enkapsulacją lub hermetyzacją, nie jest zresztą żadną nową koncepcją. Np. w poprawnym programowaniu strukturalnym korzystanie ze zmiennych globalnych, choć generalnie dopuszczalne, uważane jest jednak za swoisty brak elegancji.

Hermetyzację wymyślono już zresztą wcześniej w innej dziedzinie, a mianowicie w elektronice (choć pewnie nie tylko). Aby uniknąć nadmiernej komplikacji schematów ideowych i ułatwić projektowanie, zaczęto łączyć pewne fragmenty układów w moduły funkcjonalne, najpierw w kontekście owych układów (np. blok odchylania poziomego w telewizorze CRT), a potem bardziej ogólnie, również na potrzeby układów całkiem innych (wzmacniacze operacyjne, bramki logiczne, procesory itp.). W ten sposób powstawały podzespoły, które można było stosować w różnych urządzeniach, nie interesując się przy tym, co dokładnie mają w środku.

Drugą ważną cechą programowania obiektowego jest dziedziczenie. Korzystając z obiektów wyjściowych da się budować obiekty potomne, wyposażone w cechy owych wyjściowych obiektów oraz dodatkowo, w nowe pola i metody. Dzięki temu pewien dobrze napisany fragment kodu może uzyskać kolejne zastosowania. Ale też odwrotnie, jeśli okaże się, że zawiera on błędy, to poprawkę trzeba będzie wprowadzić tylko raz – skorygowany kod zostanie odziedziczony przez obiekty potomne. Wydaje się, że w informatyce była to koncepcja istotnie nowa, bardzo ciekawa i dająca programistom do ręki wygodne i potężne narzędzie.

Oczywiście, dziedziczenie również wymyślił kto inny i to dużo wcześniej. Bynajmniej nie mamy tu na myśli Platona, ani Arystotelesa, gdyż wbrew pochopnym sądom autora owych kliku niefortunnych zdań w *Wikipedii*, cienie w jaskini oraz dualność materii i formy odnoszą się do czegoś zupełnie innego. Wydaje się natomiast, że dziedziczenie zastosowali starożytni Rzymianie tworząc system skodyfikowanego prawa, którego używamy aż po dzień dzisiejszy. Zasady uniwersalne uszczegóławia się w konstytucji oraz w innych ustawach, dalej doprecyzowuje w rozpo-

rządzeniach, zarządzeniach oraz wyrokach sądów. Czy czegoś to nie przypomina nam, informatykom?

Zwykle, mówiąc o programowaniu obiektowym, wymienia się jeszcze dwie inne jego cechy, a mianowicie abstrakcję i polimorfizm. Pierwsza z nich w istocie sprowadza się jednak do koncepcji dziedziczenia, natomiast druga, przynajmniej na pozór wydaje się dość techniczna. Pozwala ona stosować w obiektach różnego rodzaju tę samą (odziedziczoną) metodę, której działanie jest jednak różne. Przykładowo, polecenie *wyświetl*, odnoszące się do obiektu *multimedia*, inaczej będzie realizowane dla potomnego obiektu *zdjęcie*, a inaczej dla potomnego obiektu *film*, choć w obu przypadkach będzie można je wywołać.

Polimorfizm stanowi więc przykład kontekstowego rozumienia pojęć. Suma w teorii mnogości jest czym innym niż suma w arytmetyce, choć zbieżność nazw wcale nie ma charakteru przypadkowego (arytmetyka liczb kardynalnych!) w przeciwieństwie np. do pojęcia *dystrybucja*, które w analizie funkcjonalnej jest czymś zupełnie innym niż w geometrii różniczkowej. *Moduł* liczby rzeczywistej w pierwszej chwili wydaje się z kolei być różny od modułu liczby zespolonej, jednak okazuje się dokładnie tym samym, gdy liczby rzeczywiste potraktujemy jako podzbiór liczb zespolonych. Można przypuszczać, że owo „polimorficzne myślenie" legło tak naprawdę u podstaw teorii kategorii (1945 rok) i koncepcji funktorów ko- i kontrawariantnych. Nie rozwijając tego tematu pragniemy polecić wszystkim, którzy sądzą, że to informatycy wymyślili polimorfizm, by zainteresowali się, czemu w matematyce operację brania brzegu oznacza się tym samym symbolem ∂, co różniczkowanie cząstkowe, by przypomnieli sobie, o czym mówi twierdzenie Stokes'a i co to jest kohomologia de Rahma. A może zresztą w ogóle konstruowanie polimorfizmów jest nieodłączną cechą matematycznego abstrahowania?

4. Czy ktoś jeszcze myśli tak samo jak my?

A zatem: *Nihil novi sub sole!* Wszystkie idee, które stoją za podejściem obiektowym wymyślił już wcześniej kto inny, choć – powtórzmy to raz jeszcze – nie Platon z Arystotelesem. Czy to źle? Bynajmniej, zebranie owych pomysłów i posadzenie na informatycznym gruncie jest samo w sobie doskonałym pomysłem. Co innego, gdy planuje się reeksport, albo wręcz podejrzewa, że inni inżynierowie także myślą obiektowo. No właśnie, zastanówmy się, jakby to wyglądało, jeśli rzeczywiście tak by myśleli…

Gdyby więc użyć podejścia obiektowego np. do zaprojektowania silnika spalinowego (pracującego w cyklu Otta), należałoby przykładowo stworzyć obiekt *Cylinder*, wyposażony między innymi w metodę *Zapal*. Wyobraźmy sobie teraz inżyniera, który z roztargnienia zapomni o tej metodzie. Tłok spręża mieszankę, na świecy przeskakuje iskra, a tu… nic. Nonsens, prawda? Jeżeli komuś z czytelników nie odpalił kiedyś samochód, nie przyszło mu chyba do głowy, że inżynier zapomniał zaimplementować metodę *Zapal*! Zastanawiamy się raczej nad tym, czy działa aparat zapłonowy, czy kable nie są uszkodzone, czy nie zabrakło paliwa itp. Jeśli mieszanka w cylindrze została odpowiednio sprężona, a na świecy przeskoczyła iskra, to możemy być pewni, że zapłon nastąpi. Dlaczego? Bo gwarantują to **prawa fizyki**.

Co innego oczywiście, gdy to programista modeluje silnik spalinowy. Wtedy wywołanie *Cylinder.Zapal* jest koniecznością, bo samo sprężenie oraz iskra nic nie da. W programie nie obowiązują żadne prawa, więc o wszystko trzeba zadbać samemu. Z jednej strony to miłe, bo można tworzyć wirtualne światy, których mieszkańcy potrafią np. latać bez skrzydeł lub innych podobnych instrumentów, z drugiej jednak nie można mieć pewności, że jabłko, które urwie się z jabłoni, spadnie na głowę dżentelmenowi wylegującemu się pod nią w najlepsze. Wirtualny Newton mógłby więc nie sformułować prawa powszechnego ciążenia, gdyby programista zapomniał o wywołaniu *Jabłko.Spadnij*.

Dyskusję, czy Pan Bóg wykonał świat obiektowo, a wszystko działa sprawnie wyłącznie dzięki Jego nieomylności i całkowitemu braku błędów w zrealizowanym dziele, pozostawimy na boku jako mało naukową. Zajmiemy się natomiast tym, co bardziej dostępne, a więc techniką oraz poznaniem.

Oczywiście wbrew temu, co głoszą niektórzy z przywołanych na początku studentów, metody obiektowe nie są stosowane w żadnej z dziedzin techniki poza informatyką. Ani silników spalinowych, ani domów, ani nawet układów elektronicznych (tych bez software'u) nie projektuje się obiektowo. W świecie materialnym obowiązują prawa fizyki, projektant musi się do nich stosować i żadnego wpływu na te prawa nie ma. Nie jest też tak, jak uważa część kolegów po fachu, że metody obiektowe przyjmą się gdzieś poza informatyką. Informatycy nie wynaleźli żadnego genialnego sposobu, który inni inżynierowie przyjęliby jako objawienie. Przeciwnie, pewna pokora i świadomość tego, jak młodą dziedziną techniki jest informatyka, powinny przekonać nas, byśmy nauczyli się czasem czegoś od naszych starszych braci. Ale o tym za chwilę.

5. A może świat jest relacyjny?

Każdy, kto próbował użyć technik obiektowych do modelowania jakiegoś wycinka rzeczywistości, natknął się na pewien fundamentalny problem, gdy przyszło mu zastanowić się nad składowaniem danych. Jak odwzorować obiekty w strukturę relacyjną? A może nie odwzorowywać, tylko składować je w postaci obiektowej?

Warto znów przypomnieć nieco faktów historycznych. Model relacyjny opracowany został w roku 1970 (jak wiele się wówczas zdarzyło!) przez pracującego dla firmy IBM Edgara Franka Codda ([Codd70]). Nie wszyscy mają dziś świadomość faktu, iż jedną z ważniejszych motywacji stworzenia owej koncepcji była chęć zoptymalizowania przechowywania danych pod kątem zajmowanego przez nie miejsca. W latach siedemdziesiątych pamięć była jeszcze bardzo droga, a o terabajtowych dyskach nikt nawet nie marzył, w związku z czym klucze i relacje postrzegano w dużej mierze jako sprytną metodę lepszego wykorzystania zasobów. Jaki jest jednak powód, że dziś, po czterdziestu latach, gdy terabajt dysku kosztuje niespełna 100$, nadal mówimy o kluczach i relacjach i spotykamy się na konferencjach PLOUG? Wydaje się, że nie jest to tylko tzw. „zaszłość". W istocie model relacyjny posiada jeszcze dwie inne, istotne zalety. Po pierwsze zapewnia integralność danych, przyczyniając się do ich czystości, która z kolei w oczywisty sposób przekłada się na realną użyteczność gromadzonych informacji. Po drugie, wydaje się, że jest on dla istot ludzkich dość naturalny, w przeciwieństwie do tytułowej obiektowości. Dzieje się tak dlatego, że jedną ze standardowych dróg do poznania otaczającej nas rzeczywistości (o czym więcej w następnym rozdziale) jest jej katalogowanie i porządkowanie. W ten sposób biologowie dorobili się systematyki, chemicy – układu okresowego, a fizycy – modelu standardowego cząstek elementarnych. Warto podkreślić, że klasyfikacje te powstawały jeszcze zanim sformułowano dla nich mocne podstawy teoretyczne: systematyka istniała już przed Darwinem, Mendelejew nie był świadom wyglądu zewnętrznych powłok elektronowych w atomach, zaś co do modelu standardowego, to chyba większość fizyków ma wrażenie, że stoi za nim coś więcej niż tylko grupy U(1), SU(2) i SU(3).

Dość naturalną formą katalogowania jest hierarchia, oparta de facto na relacji inkluzji, a zatem hierarchiczne bazy danych wydają nam się również czymś bardzo naturalnym. Wyjaśnia to zapewne karierę systemu Lotus Notes. Katalogowanie w bazach relacyjnych ma nieco bardziej subtelny i wyrafinowany charakter, jednak odwołuje się do znanej wcześniej koncepcji odsyłaczy do miejsc zawierających uporządkowane informacje (słowniki). To kojarzy się z ładem i porządkiem, podczas gdy odsyłacze do kolejnych odsyłaczy itd., a więc struktura sieciowa, sprawia zawsze wrażenie chaosu. Oczywiście apostołowie obiektowości mogą zarzucić nam w tym miejscu, że dziedziczenie jest bliższe hierarchiczności, aniżeli struktury relacyjne. To prawda, jednak porządkowanie pól i metod na raz sprawia wrażenie czegoś bardzo egzotycznego. Model relacyjny wy-

daje się więc tak naprawdę kompromisem pomiędzy dążeniem do uporządkowania informacji w sposób dla ludzi zrozumiały i przystępny, a chęcią lub raczej koniecznością zapewnienia tejże informacji należytej spójności i czystości.

Oczywiście, były i są podejmowane próby tworzenia obiektowych baz danych, by wspomnieć choćby *Jasmine* autorstwa Computer Associates lub możliwości, jakich dostarcza Oracle od wersji 8i. Nie popełnimy jednak chyba wielkiego błędu, jeśli stwierdzimy, że bazy obiektowe nie są stosowane na skalę przemysłową i jak na razie, raczej niewiele wskazuje na to, by miało się to zmienić. Czy chodzi tu tylko o wydajność (por. np. [Jabł03]), którą oczywiście można sukcesywnie poprawiać? Naszym zdaniem, nie.

6. A tak robią to naukowcy…

Mówi się, że podejście obiektowe jest „receptą na złożoność". Otaczająca nas rzeczywistość jest istotnie bardzo skomplikowana, więc by ją zrozumieć, opisać, a następnie poprawnie wymodelować, trzeba zastosować jakąś sensowną metodykę, a taką właśnie, jedyną i najlepszą miałoby rzekomo być podejście obiektowe.

Spróbujmy zatem krytycznie przeanalizować to stwierdzenie. Kto najlepiej zna się na badaniu, opisywaniu, bądź modelowaniu rzeczywistości? Informatycy? Nie, naukowcy! Zajmowali się tym już tysiące lat przed skonstruowaniem pierwszego komputera, zaś burzliwy rozwój przyrodoznawstwa w ostatnich wiekach, czy wręcz dekadach, doprowadził do powstania maszyn cyfrowych, nie na odwrót. Czy zatem naukowcy stosowali lub stosują metody obiektowe?

O ile biologia i chemia znane są autorowi głównie ze szkoły (nie mówiono tam nic o obiektowości), o tyle z fizyką zetknął się on nieco bliżej i może zapewnić, że nie słyszał nigdy o fizyku stosującym w swojej pracy analizę obiektową. Miejmy więc nieco pokory i nie uczmy naukowców, jak się uprawia naukę! Przeciwnie, skoro doszliśmy już w informatyce do momentu, w którym przestają nas interesować zarówno elektroniczne jak i logiczne trzewia komputera, a koncentrujemy się na poznawaniu otaczającej nas rzeczywistości, by tę odwzorować w program, to może od innych „starszych braci" dowiemy się, jak należy się do tego zabrać.

Biorąc pod lupę to, co robią uczeni, można stwierdzić, że aby w ogóle zdefiniować naukę, należy określić dwa jej aspekty: obszar zainteresowań i metodę poznawczą. Oczywiście jedno i drugie zmieniało się i zapewne będzie się zmieniać wraz z upływem czasu. Obszar zainteresowań chemii i fizyki był ongiś dość zbliżony, natomiast różne były metody. Z czasem udało się dojść do tego, że chemia to fizyka zewnętrznych powłok elektronowych (a więc doprecyzować obszar jej zainteresowań), natomiast metody badawcze obu nauk stały się do siebie bardzo podobne. Z kolei biologia zawsze interesowała się materią ożywioną, więc obszar jej zainteresowań zasadniczo się nie zmieniał – w przeciwieństwie do metod, oczywiście. Metody ewoluują, jednak ich sprecyzowanie na określonym etapie rozwoju jest czymś niezwykle ważnym. Jeżeli zastanowić się głębiej, nie ma dziś już np. czegoś takiego jak filozofia przyrody, gdyż filozofia nie oferuje żadnych metod poznawczych, które mogłyby konkurować z metodami nauk przyrodniczych. Istnieje natomiast, rzecz jasna, i ma się nieźle, filozofia przyrodoznawstwa.

W zagadnieniach informatycznych nie mamy na ogół problemu z obszarem zainteresowań – jest on dobrze określony, gdyż zawsze tworzymy oprogramowanie do czegoś konkretnego, niezależnie, czy chodzi o funkcjonowanie przedsiębiorstwa handlowego, czy loty kosmiczne. Co do metod, wygląda to już nieco gorzej – w naukach panuje na pewno większy konsensus w tym zakresie, ale to akurat można spokojnie zrzucić na karb niemowlęcego wieku naszej dyscypliny.

Pozostaje jeszcze jedna kwestia, która zasadniczo nie wchodzi w definicję żadnej z nauk, lecz w bardzo istotny sposób wyznacza rozwój każdej z nich. Mamy tu na myśli pewne ogólne idee i aprioryczne założenia, same w sobie nienaukowe (bo niefalsyfikowane, jak mówi Popper), jednak określające paradygmaty, w których nauki tkwią przez stulecia. My w informatyce również

mamy takie idee. Jedną z nich jest wiara, że każdy (nietrywialny) program da się napisać bezbłędnie. Celowo używamy słowa *wiara*, gdyż po pierwsze, nie da się tego dowieść, po drugie, nikomu się to dotychczas nie udało, po trzecie, zgodnie ze wszystkimi zasadami sztuki testowania, nikt nawet nie próbuje tego robić. A jednak wierzymy i wiara ta jest dla nas ważna. Niektórzy oczywiście wierzą również w obiektowość świata…

A w co wierzą naukowcy? W całym przyrodoznawstwie niezwykłą karierę odegrały dwie koncepcje: **atomizm** i **redukcjonizm**. Pierwsza z nich, wywodząca się jeszcze z poglądów Demokryta, pozwoliła biologom myśleć w kategoriach tkanek i komórek, by dojść wreszcie do czterech zasad: adeniny, guaniny, cytozyny i tyminy, budujących DNA. W chemii i fizyce kazała ona z kolei szukać atomów, a następnie sprawdzać, czy rzeczywiście są one *atomos* (niepodzielne) i badać dalej ich strukturę schodząc w głąb, do cząstek coraz bardziej elementarnych.

Zasada redukcjonizmu jest natomiast bardzo ważnym metodologicznym założeniem, mówiącym, że znając reguły rządzące tym, co elementarne, da się opisać układy złożone. Innymi słowy, mając do dyspozycji równanie Diraca, określające zachowanie elektronu, jesteśmy w stanie np. opisać budowę atomu sodu i wyjaśnić, dlaczego jego pary świecą na żółto. Idąc dalej, możemy modelować zewnętrzne powłoki atomów, a zatem rozumieć reakcje chemiczne, a to z kolei pozwala nam zmierzyć się z fenomenem, jakim są procesy zachodzące w żywych komórkach.

Oczywiście, nie jest naszym celem zajmowanie się tu filozofią nauki, lecz raczej uzmysłowienie informatykom, że ktoś już przed nimi wymyślił, jak analizować i opisywać złożoność i osiągnął przy tym, nota bene, całkiem niezłe rezultaty. Czy zatem ów sposób myślenia przyrodników dałoby się zastosować podczas opisu np. funkcjonowania przedsiębiorstwa? Spróbujmy.

7. Przedsiębiorstwo potraktowane naukowo

Zaczynamy tradycyjnie, od analizy procesów. Przyrodnicy nazywają je zjawiskami. Opisujemy procesy czysto fenomenologicznie, głównie po to, by zorientować się, z czym będziemy mieli do czynienia. Mniej więcej podobną pracę wykonują fizycy doświadczalni – badają, opisują, mierzą. Metodyka jest tu dowolna – jeżeli ktoś lubi, może użyć np. języka UML. Następnie staramy się zidentyfikować atomy, a więc najmniejsze elementy, z jakimi przyjdzie nam się zetknąć oraz układy owych atomów (molekuły, komórki, tkanki – nomenklatura wedle upodobania). Atomem może być cały Dział Handlowy, jeśli nie ma w nim żadnej specjalizacji, mogą być jego wydziały, a nawet poszczególni pracownicy. Atomami mogą być dokumenty lub elementy składowe owych dokumentów (np. pozycje ofert), w zależności od sytuacji. Na samym końcu należałoby opisać interakcje wspomnianych atomów i zidentyfikować reguły nimi rządzące, a więc tzw. fundamentalne prawa natury. Tym zajmuje się fizyka teoretyczna. Zgodnie z zasadą redukcjonizmu wystarczą one do scharakteryzowania funkcjonowania całego przedsiębiorstwa. Proste? Na pewno nie. Przynajmniej w praktyce. Ale chyba jednak znacznie bardziej naturalne, niż to co informatycy robią obecnie!

Teraz jeszcze potrzebny byłby język programowania, który pasowałby do takiego „naukowego" podejścia. Musiałby on oczywiście umożliwiać stworzenie repozytorium atomów oraz praw nimi rządzących. To jednak nie wszystko. Naszym celem jako programistów nie jest przecież zbudowanie opisu funkcjonowania przedsiębiorstwa, ale programu wspomagającego pracę zatrudnionych w nim osób. Chodzi zatem o to, by np. pismo sklasyfikowane przez sekretarkę jako zapytanie ofertowe, skierowane zostało do działu handlowego, by następnie nasza aplikacja „skłoniła" któregoś z handlowców do przygotowania oferty, pomogła mu ją stworzyć itd. Mówiąc krótko, chcielibyśmy we wspomnianym już języku programowania zbudować pewne urządzenie, wykorzystujące odkryte uprzednio reguły. Oznacza to, jak łatwo zauważyć, sprowadzenie zagadnienia informatycznego do typowego zagadnienia technicznego – typowego, to znaczy analogicznego do zagadnień spotykanych we wszystkich innych dziedzinach techniki. Mamy więc wreszcie

to, czego wielu z nas tak bardzo pragnęło! Możemy stosować metody takie, jak inni koledzy inżynierowie, a więc sprawdzone przez stulecia lub może nawet tysiąclecia, nie zaś wymyślać wciąż nowe, cudowne recepty, a niepowodzenia projektów tłumaczyć tym, że przecież informatyka jest jeszcze młodziutka i dopiero w przyszłości zapewne okrzepnie. Jednym się to spodoba, innym pewnie nie…

A czymże powinien być ów cudowny język programowania? Jeżeli skojarzenia czytelnika są tu podobne do skojarzeń autora, to zapewne przyszły mu na myśl gry komputerowe. One bowiem mają zwykle swoją własną fizykę rządzącą wirtualnym światem, zaś realizuje się ją poprzez odpowiedni motor, dzięki któremu programista nie musi za każdym razem dbać o to, by bohater rażony, powiedzmy, granatem atomowym wyparował, a nie zaczął gwizdać *Serce Słowianki* (ukłony dla Marii Czubaszek). Oczywiście nie wiemy, czy dwa kolejne życia zapewni naszemu bohaterowi automatycznie ów motor, czy też inne zabiegi programistyczne. Wszystko zależy od tego, gdzie twórcy gry wyznaczyli subtelną granicę między standardowymi „prawami fizyki" a ingerencją nadzwyczajną, powszechnie nazywaną cudem.

Wygląda więc na to, że koncepcję, którą lansujemy w tym artykule, wymyślili już wcześniej twórcy gier komputerowych (włączając w to pojęcie również wszelkiego rodzaju symulatory, bynajmniej nie służące rozrywce). Należałoby więc teraz zrobić krok dalej, a więc rzeczywiście skonstruować ów specjalny język, który po skompilowaniu kodu źródłowego sam tworzyłby motor fizyki, a następnie zacząć używać owego języka (lub może nowego paradygmatu) do tworzenia szerokiej gamy software'u. Nasz przykład z przedsiębiorstwem pozostaje jak najbardziej aktualny.

Oczywiście autor może nie być świadom wszystkich pomysłów zmierzających w kierunku owego nowego paradygmatu. Nie ma jednak wątpliwości, że ewentualne próby nie wyszły poza sferę prac naukowych, zaś zastosowanie jego elementów w obszarze wspomnianych gier komputerowych miało charakter tyleż oczywisty – skoro tworzymy wirtualny świat, najprościej zbudować jego wirtualną fizykę – co nieświadomy, ponieważ chyba nie próbuje się iść dalej. A szkoda, tym bardziej, że niejako równolegle rozwija się inne podejście, które wykazuje duże podobieństwo do postulowanego tutaj paradygmatu.

8. Semantyka i ontologia

No właśnie, w ostatnich latach coraz większym zainteresowaniem cieszą się pewne koncepcje, które w swoich nazwach mają słowa *semantyka* i *ontologia*. Aby przekonać się o tym wystarczy spojrzeć choćby na materiały z kilku poprzednich konferencji PLOUG ([JBCM06], [FaJę07], [JęBą07], [Morz09], [BąJę09]).

Jak się wydaje, główną motywacją dla podjęcia badań, których dotyczą między innymi wyżej wymienione publikacje, a także dla rozwoju odpowiednich technologii, jest potrzeba jakiegoś sensownego ogarnięcia ogromu informacji zgromadzonych w Internecie. Prawdopodobnie jedynym rozwiązaniem tego problemu byłoby coś, co można by określić mianem przetwarzania danych ze zrozumieniem (przez maszyny oczywiście). Stąd właśnie termin *semantyka*, odwołujący się do znaczenia słów, którym to znaczeniem, a nie samymi ciągami zer i jedynek, powinny zajmować się komputery. Przykładem bardzo prostego języka służącego do opisu semantycznych powiązań między danymi może być RDF (*Resource Description Framework*), pozwalający formułować stwierdzenia składające się z trzech elementów: podmiotu, predykatu i obiektu (por. np. [Morz09]). Wystarczy to zapewne, by nauczyć komputer, że *Ala ma kota*, ale nie wystarczy, by napisać, nawet na ten sam temat, krótki traktat filozoficzny. Oczywiście, z całym szacunkiem dla Ali i jej kota, sprawny system zarządzania bazą danych, wykorzystujący podejście semantyczne, stanowiłby duży krok naprzód w porównaniu z powszechnie wykorzystywanym obecnie modelem relacyjnym. Z drugiej strony, apetyt rośnie w miarę jedzenia, więc pokusa, by wyposażyć maszy-

nę w możliwości wnioskowania, jest silna. I w ten sposób dochodzimy do drugiego z terminów, *ontologia*, jak się wydaje wyjątkowo nieszczęśliwego.

Ontologia to nauka o bycie, drugi obok epistemologii, czyli teorii poznania, dział tradycyjnej filozofii. W istocie, chcąc sensownie zajmować się znaczeniem słów, trzeba wziąć pod uwagę cały ich kontekst, a więc dziedzinę, w obrębie której występują. Dziedzina ta to niby zbiór bytów, i stąd słowo ontologia, rozumiane nie jako nauka, ale struktura zawierająca pojęcia i powiązania między nimi, a także reguły wnioskowania, pozwalające na odkrywanie nowych powiązań. Wyznawcy mówią o ontologiach botaniki, mikroekonomii, elektroniki itp.

Gdy się głębiej zastanowić, ontologia okazuje się po prostu systemem dedukcyjnym – jak wiadomo, mamy w nim do dyspozycji pojęcia pierwotne i aksjomaty (a więc powiązania między owymi pojęciami), a korzystając z nich możemy definiować kolejne pojęcia oraz formułować i udowadniać twierdzenia na ich temat. Tak więc znowu nic nowego, choć świadome unikanie terminu *system dedukcyjny* wydaje się być wyrazem pewnego uzasadnionego lęku przed niepowodzeniem całego przedsięwzięcia, o czym więcej w kolejnym paragrafie.

Do opisu owych ontologii służyć mają języki takie jak RDFS (*Resource Description Framework Schema*), OWL (*Web Ontology Language*), czy SWRL (*Semantic Web Rule Language*), pozwalający budować reguły w postaci implikacji *jeżeli poprzednik to następnik* ([JBCM06]), a więc inaczej mówiąc, formułować twierdzenia (z założenia wynika teza). Technicznym problemem jest tu oczywiście złożoność obliczeniowa ([JBCM06], [Morz09]), natomiast o problemach fundamentalnych piszemy w następnym paragrafie.

Sam pomysł jest ciekawy i dość naturalny w kontekście całego dotychczasowego rozwoju informatyki. Wydaje się więc, że wystarczyłoby pójść o krok dalej i rozbudować wspomniane języki tak, aby nie tylko dało się w nich opisywać pewną rzeczywistość, ale również tworzyć oprogramowanie w jakimś sensie ją wspomagające. Chciałoby się zatem rozszerzyć zakres stosowalności języków, a używając metafory – przejść od nauki do techniki, od odkrycia fal elektromagnetycznych do skonstruowania radia.

9. Ograniczenia

Opisane powyżej koncepcje ontologiczne skłaniają niektórych swoich głosicieli do przekonania, że oto znowu informatycy odkryli jakąś fenomenalną metodę opisywania rzeczywistości, którą niczym objawienie przyjmą przedstawiciele nauk przyrodniczych. Ci ostatnie otrzymaliby więc rewelacyjny formalny język, którego wcześniej sami nie potrafili się dorobić, a który pomógłby im precyzyjnie opisać cały naukowy dorobek. Przestrzegamy przed takim brakiem pokory! Naukowcy wypracowywali swój język przez stulecia i z jakichś powodów jest on właśnie taki, jaki jest, a jeśli ewoluuje, to raczej nie w kierunku wyznaczanym przez informatyków. Dlaczego? Ponieważ język nauk (przyrodniczych) wcale nie jest precyzyjny, a stopień owej nieprecyzyjności bywa różny w różnych dziedzinach. Mówi się żartobliwie, że chemicy śmieją się z biologów, fizycy z chemików, matematycy z fizyków, a logicy z matematyków. Z logików śmieją się z kolei filozofowie, a z filozofów to się wszyscy śmieją. Tak przy okazji, kto będzie śmiać się z informatyków?

Fakt pozostaje jednak faktem, że nawet królowej nauk nie udało się w pełni sformalizować! Próbował tego David Hilbert, lecz jego słynny program aksjomatyzacji matematyki legł w gruzach, gdy w 1931 roku Kurt Gödel udowodnił swoje słynne dwa twierdzenia. Przypomnijmy, bez wnikania w szczegóły, że pierwsze z nich mówi o niemożności dowiedzenia wewnętrznej niesprzeczności (już niezbyt skomplikowanych) systemów dedukcyjnych, zaś drugie o tym, że w takich systemach pojawiają się zdania, których prawdziwości (ani fałszywości) nie można udowodnić. Skoro więc da się w sposób ścisły pokazać, że sformalizowanie matematyki nie jest możliwe,

nie polecamy informatykom zabierać się za takie zadanie. Podobnie zresztą jak za formalizowanie fizyki, chemii, czy biologii.

Jak ma się to do naszej propozycji naukowego opisu funkcjonowania przedsiębiorstwa i budowy systemu informatycznego funkcjonującego w oparciu o taki opis? Każdy, kto choćby przez chwilę pracował w charakterze analityka, uczestniczył zapewne w następującym dialogu:

– Czy tak się to właśnie u was robi?
– Dokładnie tak.
– Zawsze?
– Oczywiście, że zawsze.
– A co, jeżeli…
– A, wtedy robi się to trochę inaczej, ale to nie zdarza się często.
– A jeżeli…
– No, to wtedy trzeba zrobić to jeszcze inaczej, ale to przecież oczywiste, wszyscy to wiedzą.

Samo życie, prawda? Wydaje się, że nasz cudowny język powinien być wyposażony w dobrą obsługę wyjątków i to nie tego typu, jakie realizuje się przy pomocy klauzuli *try*.

Drugim poważnym problemem jest sposób, w jaki stosuje się prawa fizyki do opisu konkretnych zjawisk, a więc również podczas konstruowania urządzeń wykorzystujących te zjawiska. I tak, prawa fundamentalne z założenia nie powinny mieć żadnych ograniczeń w zakresie obowiązywania. W rzeczywistości jest nieco inaczej, gdyż teorie fizyczne zawsze posiadają pewne obszary zastosowań. Niekiedy widać je dopiero z perspektywy teorii nowszych i lepszych. Przykładowo, ograniczenia teorii grawitacji Newtona dostrzega się z perspektywy ogólnej teorii względności, dla której ta pierwsza stanowi liniowe przybliżenie. Czasami jednak teorie same ukazują swoje ograniczenia prowadząc do wniosków w oczywisty sposób absurdalnych. Tak jest chociażby z elektrodynamiką Maxwella, termodynamiką klasyczną, czy nawet mechaniką kwantową, która, jako jedyna z teorii fizycznych, wymaga tzw. (filozoficznej) interpretacji.

Jakby jednak nie było, prawa uważane za fundamentalne powinny obowiązywać zawsze i bezwzględnie, niemal z definicji. Ich rzeczywiste stosowanie napotyka jednak na realne problemy, bo o ile przywołana wcześniej zasada redukcjonizmu każe wierzyć, że dysponując równaniem Diraca wyznaczymy poziomy energetyczne sodu, o tyle praktyczne obliczenia wymagać będą użycia różnych metod przybliżonych. Zaledwie jedenastoelektronowy atom jest już po prostu bardzo skomplikowany. Warto uzmysłowić sobie, że w sposób analityczny udało się „rozwiązać" jedynie atom wodoru!

W rzeczywistości budujemy więc pewne modele, które niekiedy wydają się bardzo odległe od rzeczywistości, lecz oddają pewne jej cechy, istotne z punktu widzenia przeprowadzanych rachunków. Na przykład, by pokazać ekspansję Wszechświata założono niegdyś (Aleksander Friedmann), że jest on obecnie równomiernie wypełniony pyłem. Czym są ziarna tego pyłu? Gwiazdami? Wystarczy spojrzeć w niebo, by dostrzec, że to bzdura. Galaktykami? Gromadami galaktyk? Nie! Ale w ogóle nie o to chodziło. Ktoś naszych rozmiarów uprawiając naukę nie powinien zapewne nigdy stwierdzać, że gwiazda to ziarno pyłu, nie mówiąc już o galaktyce lub gromadzie galaktyk! A jednak, owo osobliwe założenie modelowe pozwoliło osiągnąć bardzo ciekawe wyniki, których zasadniczej treści nie podważa się do dziś, chociaż minęło już prawie sto lat, podczas których wielu badaczy zdołało wykonać znacznie dokładniejsze obliczenia. Uproszczenia modelowe są w fizyce wszechobecne i niekiedy trudno nawet ustalić, czy w istocie odnoszą się one do opisu konkretnych zjawisk, czy raczej do sformułowania praw podstawowych. Jak wyglądałaby ta dziedzina wiedzy, gdyby pozbawiono ją takich abstrakcji jak punkt materialny, ciało doskonale czarne, bryła sztywna, ciecz nielepka, zdarzenie elementarne itd. itd.?

Uproszczenia modelowe wykorzystywane są oczywiście również przez inżynierów podczas projektowania całkiem realnych urządzeń, a fakt, iż urządzenia te działają, jest najlepszym spraw-

dzianem stosowanych metod, choć np. schemat zastępczy tranzystora na pierwszy rzut oka bardzo odbiega od tego, czym tranzystor jest w rzeczywistości.

W kontekście rozważanego wcześniej przykładu opisu funkcjonowania przedsiębiorstwa oraz budowy systemu informatycznego wspomagającego jego pracę również nie unikniemy uproszczeń modelowych. Naiwnością byłoby sądzić, że odwzorujemy w system całą rzeczywistość gospodarczą ze wszystkimi jej niuansami. Ważne natomiast jest, aby przybliżenia, które robimy, miały zawsze charakter kontrolowany, a więc by wiadomo było, jakiego rodzaju i jak duże błędy popełniamy, gdy je stosujemy. Nie musimy nawet dodawać, że te same uproszczenia w pewnych sytuacjach będą uprawnione, w innych zaś nie. Na przykład, projektując wózek widłowy nikt nie próbuje nadać mu opływowego kształtu, bo przy prędkościach, które wózek taki rozwija, opór powietrza jest zaniedbywalny. Co innego, gdy projektuje się Ferrari… Podobnie jest z oprogramowaniem. Pewien program służący do fakturowania pozwalał wystawić wiele korekt do jednej faktury, jednak nie umożliwiał korygowania samych korekt. Księgowa argumentowała, że program jest ułomny, bo prawo pozwala korygować korekty, korekty korekt itd. w nieskończoność. Zapytana, jak często wystawia korekty do korekt, przyznała że czyni to kilka razy w roku, ale w zasadzie w ogóle nie musi tego robić. Nawet jednak gdyby musiała, wypisanie paru takich dokumentów ręcznie w ciągu roku nie byłoby zapewne specjalnym problemem. Co innego, gdyby chodziło o same korekty do faktur, a wypisywałoby się ich choćby dwie lub trzy dziennie…

10. Co dalej?

Warto uzmysłowić sobie, że to analiza i projektowanie obiektowe, bądź jak chcą niektórzy, szerzej, podejście obiektowe, czy myślenie obiektowe, są pochodnymi programowania obiektowego, nie zaś jakąś głęboką filozofią, z której programowanie owo się wywodzi. Określone uwarunkowania techniczne skłoniły ludzi do stworzenia pewnych metod budowania kodu, które to metody okazały się zresztą bardzo skuteczne, zwłaszcza podczas konstruowania interfejsu użytkownika. Wraz z rozwojem techniki przychodzi czas na zmiany. Nie warto wmawiać sobie, że obiektowość jest cechą świata, bądź choćby naszego naturalnego sposobu jego postrzegania lub poznawania. Staraliśmy się przekonać czytelnika, że nie jest. Cały dorobek nauk przyrodniczych, a także osiągnięcia innych niż informatyka dziedzin techniki zdają się świadczyć o tym, że podejście obiektowe stanowi raczej ślepek zaułek, w który nieopatrznie zabrnęliśmy. Należy więc go jak najszybciej opuścić i – korzystając z możliwości, jakie daje rozwój technologii informatycznych – pójść w bardziej sprawdzonym kierunku.

Czy proponowana w tym artykule koncepcja (*paradygmat naukowy*?) sprawdzi się, tego oczywiście nie wiemy. Mimo rosnącego zainteresowania „ontologiami" dziś jeszcze może się ona wydawać kompletną abstrakcją, jednak nie wiadomo, czy jutro nie zostanie uznana za coś najbardziej naturalnego w informatycznym świecie. Tak już bywało. Jeszcze u początku lat osiemdziesiątych ubiegłego wieku, kiedy w Krakowie stała potężna jak na owe czasy i nasz kraj maszyna firmy CDC, Cyber 70, niektórzy żartowali, że w przyszłości zapewne będą mieć takie na biurkach. Żartowali, choć nie do końca w to wierzyli. Dziś mocniejsze komputery noszą w kieszeni. Z drugiej jednak strony, postęp nie zawsze zaskakuje nas in plus. W latach pięćdziesiątych prognozowano, że nie miną dwie dekady, a energia elektryczna produkowana będzie w elektrowniach termojądrowych, nie mówiąc już o tym, że wiek XXI kojarzył się wówczas wszystkim z powszechną robotyzacją i masową turystyką na Księżyc. I co? Elektrowni termojądrowych jak nie było, tak nie ma. Są za to wszelkie szanse na to, że nasze czasy określi ktoś kiedyś jako epokę wojen religijnych, terroryzmu i powszechnego lęku przed pandemiami. Zaiste trudno przewidzieć przyszłość, jednakże warto nakreślać i promować pewne technologiczne wizje. Może wtedy niektóre z nich się zmaterializują…

Bibliografia

[BąJę09] Bąk, J., Jędrzejek, C.: Semantic Web – technologie, zastosowania, rozwój, XV Konferencja PLOUG, Kościelisko, październik, 2009, ss. 236-246, ISSN 1641-2117.

[CoYo94] Coad P., Yourdon E.: Analiza obiektowa, Oficyna Wydawnicza READ ME 1994, ISBN 83-85769-17-X.

[Codd70] Codd, E.F.: A Relational Model of Data for Large Shared Data Banks, Communications of the ACM, Vol. 13, No 6, June 1970, ss. 377-387, ISSN 0001-0782.

[DKK98] Dumnicki, R., Kasprzyk A., Kozłowski, M.: Analiza i projektowanie obiektowe, Wydawnictwo Helion 1998, ISBN 83-7197-052-8.

[FaJę07] Falkowski M., Jędrzejek C.: Budowanie schematów relacyjnych dla danych OWL z wykorzystaniem silników, XIII Konferencja PLOUG, Kościelisko, październik, 2007, ss. 303-312, ISSN 1641-21-17.

[Jabł03] Jabłoński, B.: Badanie wydajności Oracle 9i w kontekście modelu obiektowego, IX Konferencja PLOUG, Kościelisko, październik, 2003, ss. 70-85, ISSN 1641-21-17.

[Jasz97] Jaszkiewicz, A.: Inżynieria oprogramowania, Wydawnictwo Helion 1997, ISBN 83-7197-007-2.

[JBCM06] Jędrzejek C., Bąk J., Cybulka J., Martinek, J.: Aspekty narzędziowe w projekcie systemu do analizowania naruszeń prawa, XII Konferencja PLOUG, Kościelisko, październik, 2006, ss. 273-286, ISSN 1641-21-17.

[JęBą07] Jędrzejek C., Bąk J.: Wnioskowanie hybrydowe w relacyjnej bazie danych, wykorzystujące ontologię OWL wzbogaconą regułami języka SWRL, XIII Konferencja PLOUG, Kościelisko, październik, 2007, ss. 287-300, ISSN 1641-2117.

[Morz09] Morzy M.: Semantic Technologies, czyli Oracle i Web 3.0, XV Konferencja PLOUG, Kościelisko, październik, 2009, ss. 30-43, ISSN 1641-2117.

[MaOd97] Martin, J., Odell J.J.: Podstawy metod obiektowych, Wydawnictwa Naukowo-Techniczne 1997, ISBN 83-204-2116-0.

[Tayl94] Taylor, D.A.: Technika obiektowa, Wydawnictwo Helion 1994, ISBN 83-85701-31-1.