PRACE NAUKOWE UNIWERSYTETU EKONOMICZNEGO WE WROCŁAWIU nr 119
Informatyka Ekonomiczna 18 2010

Cezary Hołub
Uniwersytet Ekonomiczny we Wrocławiu

ROZWIĄZANIA ARCHITEKTONICZNE AOP W BUDOWIE SYSTEMÓW TYPU GRID

Streszczenie: W obecnie budowanych systemach informatycznych przetwarzanie gridowe zyskuje coraz bardziej na znaczeniu. Sposoby budowy oraz architektura systemów grid ciągle ewoluują. Nieustannie poszukuje się w miarę uniwersalnych rozwiązań zwiększających niezawodność, przejrzystość i prędkość działania tych systemów. Jednym z pomysłów jest wprowadzenie rozwiązań bazujących na programowaniu aspektowym AOP i rozdziale zagadnień. System grid, będący odmianą systemu rozproszonego, realizuje pewne funkcjonalności w swoich węzłach. Architektura i rozwiązania AOP umożliwiają wyodrębnienie tych funkcjonalności i zamknięcie ich w dany aspekt, a następnie ich realizację w jednym miejscu. Ten artykuł jest próbą ukazania możliwych zastosowań AOP w systemach grid oraz pokazaniu zalet takiej realizacji; spróbujemy także znaleźć wspólną płaszczyznę dla tych technologii.

Słowa kluczowe: AOP, grid, architektura, projektowanie.

1. Wstęp

Chcielibyśmy, aby systemy informatyczne były coraz bardziej wydajne, coraz mniej awaryjne, coraz prostsze w konstruowaniu i utrzymywaniu, a przy tym rozwiązywały coraz to bardziej złożone problemy ze świata rzeczywistego. Od dawna można obserwować tendencje do osiągania przez programy komputerowe coraz większych rozmiarów. Ze wzrostem tym wiąże się przyrost złożoności i skomplikowania powstających systemów informatycznych.

Organizacje traktujące technologie informatyczne jako kluczowe narzędzie warunkujące sprawne zarządzanie powinny być świadome współczesnych wyzwań dotyczących m.in. terminowego dostarczania odpowiednich informacji bądź wiedzy czy też korzystania z wielu źródeł informacyjnych. Chodzi bowiem o efektywne wspomaganie procesów biznesowych z uwzględnieniem optymalizacji przepływu informacji oraz pozyskiwania niezbędnej wiedzy. Naturalne jest wobec tego oczekiwanie na rozwiązania, które wychodzą naprzeciw tym wyzwaniom. Jednym z takich rozwiązań jest system typu grid.

System typu grid powinien stanowić znaczący krok w rozwoju obsługi informacyjnej współczesnych organizacji – działających przecież w bardzo zróżnicowanych warunkach: od takich, które dysponują ograniczonymi zasobami, po instytucje globalne, operujące często w przestrzeni wirtualnej na prawie nieograniczonych infrastrukturach informacyjnych.

Głównym celem artykułu będzie pokazanie, że niektóre cechy podejścia AOP (*Aspect Oriented Programming*), a szczególnie rozwiązania architektoniczne, dobrze sprawdzają się w przypadku tworzenia systemów typu grid. Część pierwszą stanowi wstęp. W części drugiej zostanie przedstawiona krótka specyfikacja programowania aspektowego. Część trzecia opisuje pokrótce istotę i znaczenie przetwarzania gridowego. Część następna, będąca główną częścią artykułu, pokazuje możliwe rozwiązania architektoniczne cechujące podejście AOP w budowie systemów typu grid. Artykuł kończy się podsumowaniem.

2. Specyfika programowania aspektowego

Wiele realizowanych zagadnień w systemie informatycznym pociąga za sobą w praktyce potrzebę realizacji zagadnień pobocznych. Można je traktować jako niefunkcjonalne wymagania względem systemu, których implementacja powoduje rozsianie kodu w wielu miejscach modelu funkcjonalnego. Najczęściej są to kwestie logowania, bezpieczeństwa, spójności transakcyjnej, autoryzacji, synchronizacji wielowątkowej i wiele innych. Jest to zjawisko wynikające ze złożoności wymagań klienta. Zagadnienia te są w dużym stopniu rozłączne między sobą pod względem funkcjonalnym. Aby je zrealizować, programista musi poprzeplatać ich implementacje, co czyni kod mniej czytelnym, bardziej podatnym na błędy, trudniejszym w modyfikacji.

Programowanie aspektowe zapobiega tym negatywnym skutkom, oddzielając fizycznie kod każdego zagadnienia poprzez umieszczenie ich w oddzielnych aspektach i logiczne zdefiniowanie punktów interakcji pomiędzy nimi.

W systemach o niskim stopniu modularyzacji wszystkie potencjalne zmiany w programie wiążą się z ogromnym ryzykiem wygenerowania nowych błędów, nie mówiąc o kosztach takich zmian. Znaczna większość powstającego obecnie oprogramowania implementowana jest za pomocą języków obiektowych. Podejście obiektowe pomaga lepiej zrozumieć rozpatrywany problem (obiekty odzwierciedlają elementy świata rzeczywistego). Ułatwia też powtórne wykorzystywanie stworzonych elementów i ich przystosowywanie do nowych potrzeb. Niestety, takie podejście ma również wady. W przypadku dużych aplikacji powstały kod zwykle nie jest zbyt przejrzysty i do końca zrozumiały. Na bazie programowania proceduralnego i obiektowego, zapewniającego już dużą modularyzacje, powstała i jest cały czas rozwijana metodologia programowania aspektowego. Ma ona na celu udoskonalenie tzw. rozdzielenia zagadnień (*separation of concerns*), czyli rozdzie-

lenia pewnych aspektów funkcjonalności programu (np. synchronizacji dostępu do zasobów, śledzenia przebiegu programu) na osobne, niezależne moduły. Pożądane jest przy tym, aby modularyzacja systemu bardziej odzwierciedlała sposób, w jaki myślimy o problemie, niż metodę narzucaną przez narzędzia informatyczne. W przypadku złożonych systemów programowanie obiektowe nie pozwala na modularyzacje wszystkich zagadnień systemu. Węzły systemu grid doskonale nadają się na realizację w nich zamkniętych funkcjonalności. Pewne problemy zawsze będą przecinały granice (*crosscutting*) innych modułów. Ideą programowania aspektowego jest dostarczenie mechanizmów pozwalających na pełniejszą modularyzację systemu. Ma ono uprościć kod, zwiększyć prędkość jego powstawania, uczynić go łatwiejszym do zrozumienia, rozwoju, konserwacji i ponownego użycia. Zmodularyzowane zagadnienia przecinające (*crosscutting concerns*) nazywane są aspektami.

Za głównego twórcę AOP[1] (*Aspect Oriented Programming*) uznaje się profesora G. Kiczalesa z uniwersytetu w Vancouver, którego prace w końcowych latach dziewięćdziesiątych przyczyniły się do powstania tej metody i sformułowania jej zasad. Po raz pierwszy użył on terminu *Aspect Oriented Programming* w roku 1996 podczas swojej pracy w firmie Xerox Corporation. Metoda ta spotyka się z dużym zainteresowaniem. Rozpoczęto wiele badań nad jej zastosowaniem w różnych dziedzinach, organizowano konferencje jej poświęcone. Artykuł z roku 1997 „Aspect-Oriented Programming”[2] napisany przez G. Kiczalesa jest uważany za pierwszą publikacje o programowaniu aspektowym.

Najważniejsze pojęcia związane z programowaniem aspektowym to:

- **aspekt** (*aspect*) – aspekty zdefiniowano jako modularne jednostki przecinające implementacje,
- **zagadnienie** (*concern*) – jest to kwestia lub problem, które muszą zostać rozwiązane, aby można było osiągnąć cel systemu,
- **zagadnienie przecinające** (*crosscutting concern*) – jest to zagadnienie, którego reprezentacja jest rozproszona wzdłuż reprezentacji innych zagadnień w hierarchicznie dekomponowanym systemie. Istnienie relacji przecinania pomiędzy zagadnieniami może być zależne od wyboru dominującego kryterium dekompozycji. Na tym gruncie wyrosły zarówno modularyzacja, jak i metody obiektowe.

Jednym z rozwiązań pozwalających na wykorzystanie w praktyce idei programowania aspektowego jest język programowania AspectJ. Stanowi on uniwersalne aspektowe rozszerzenie języka Java, tzn. że każdy program Javy jest jednocześnie poprawnym programem języka AspectJ. Nie modyfikuje on żadnej konstrukcji tego języka, a dodaje nowe – przede wszystkim pojęcie aspektu. Aspekt jest specyficznym rodzajem klasy, który razem z nią pozwala zmodularyzować program. AspectJ jest pełnym językiem programowania: oprócz składni posiada także włas-

[1] http://en.wikipedia.org/wiki/Aspect-oriented_programming.

[2] http://www.parc.com/research/projects/aspectj/downloads/ECOOP1997-AOP.pdf Kiczales G.: „Aspect-Oriented Programming”.

ne narzędzia – kompilator i debugger. AspectJ, wywodząc się z Javy, przejął po niej bardzo ważną cechę, a mianowicie przenośność kodu i niezależność od platform sprzętowo-programowych. Cecha ta jest szczególnie pożądana w realizacji systemu grid w środowisku heterogenicznym.

3. Istota i znaczenie przetwarzania gridowego

Przetwarzanie siatkowe (*grid computing*) jest nowoczesną koncepcją wykorzystania rozproszonych zasobów komputerowych jako logicznej jednostki przetwarzania danych. Ogromny sukces zastosowań Internetu w dziedzinie globalnego upowszechniania informacji spowodował wzrost zainteresowania wykorzystaniem jego infrastruktury jako platformy dla realizacji aplikacji rozproszonych. W ciągu ostatnich kilkunastu lat zaproponowano szereg technologii umożliwiających implementację komponentowych aplikacji rozproszonych, których elementy mogą być rozlokowane na wielu heterogenicznych węzłach połączonych siecią typu internet lub intranet: CORBA, DCOM, Enterprise JavaBeans (EJB), WebServices. Aplikacje realizowane z pomocą tych technologii składają się zwykle z pojedynczego modułu klienta, pracującego na komputerze użytkownika końcowego i odpowiadającego za obsługę graficznego interfejsu użytkownika oraz za sterowanie przetwarzaniem, a także z wielu modułów zdalnych, osadzonych na serwerach aplikacji, odpowiadających za realizację właściwego przetwarzania danych. Charakterystyczne różnice między wspomnianymi technologiami sprowadzają się do zakresu wspieranych języków programowania (np. EJB współpracuje wyłącznie z językiem Java), platform systemu operacyjnego (np. DCOM przeznaczony jest dla środowisk Microsoft Windows) oraz stosowanych protokołów komunikacji klienta z modułami zdalnymi (np. HTTP w WebServices, natomiast CORBA i EJB korzystają z IIOP).

Pewnego rodzaju wadą wyżej wymienionych technologii budowy rozproszonych aplikacji komponentowych jest sztywność powiązania zdalnych modułów ze sprzętem lub serwerami aplikacji, do których zostały przydzielone. Osoby wdrażające rozproszone systemy komponentowe podejmują wiążące decyzje o fizycznej lokalizacji wszystkich elementów systemu. Działanie takie powoduje, że każdy system aplikacyjny posługuje się zbiorem dedykowanych komputerów, nazywanym często „wyspą przetwarzania danych" (*island of computing*), a zasoby sprzętowe przedsiębiorstwa są podzielone (najczęściej rozłącznie) pomiędzy posiadane systemy aplikacyjne. W codziennej eksploatacji może się zdarzać, że dochodzi do przeciążenia zasobów wykorzystywanych przez jeden system aplikacyjny, podczas gdy pozostałe systemy nie są w ogóle wykorzystywane przez użytkowników.

Problem sztywnego przydziału oprogramowania do sprzętu może zostać złagodzony dzięki zastosowaniu koncepcji przetwarzania *grid computing* (przetwarzanie siatkowe, przetwarzanie gridowe, przetwarzanie sieciowe). *Grid computing* postuluje, aby traktować wszystkie posiadane zasoby sprzętowe jako jeden wielki kom-

puter wirtualny, zdolny wykonywać wszystkie dotychczasowe aplikacje, automatycznie alokując je do poszczególnych maszyn w taki sposób, aby równoważyć obciążenie maszyn oraz redukować wpływ awarii na dostępność systemów aplikacji. Najczęściej mówi się o *computational grids*, które wirtualizują zasoby obliczeniowe (procesory, pamięci operacyjne), oraz o *data grids*, które wirtualizują pamięć masową (dyski twarde, napędy CD/DVD, streamery)[3].

4. Rozwiązania architektoniczne AOP w budowie systemów typu grid

Kluczowe, ze względu na funkcjonowanie systemu gridowego, wydają się być trzy jego elementy niezbędne do sprawnego działania całego systemu: sieć o dużej przepustowości, oprogramowanie warstwy pośredniczącej (*middleware*) oraz stosowane w takim układzie standardy. Stworzenie programu działającego w technologii *grid computing*, wymaga sporego doświadczenia z zakresu tworzenia oprogramowania pośredniczącego (*middleware*) – jego rola jest podobna do roli warstwy sieciowej systemu operacyjnego; aplikacja, która chce przesłać dane po sieci, nie musi znać szczegółów dotyczących konkretnych podzespołów tej sieci. W przetwarzaniu gridowym oprogramowanie to nie tylko „maskuje" strukturę sieci, ale także ma możliwość obsługi innych architektur, optymalizując czas przetwarzania całej sieci.

Głównym priorytetem przy projektowaniu systemu grid (a dokładnie jego części *middleware*) z użyciem AOP jest znalezienie i izolacja wspólnych zagadnień. W czasie projektowania będziemy mogli również skorzystać z wzorców projektowych[4] (*design patterns*) po wcześniejszej analizie miejsc możliwych wystąpień wzorców. Zalety stosowania wzorców projektowych w budowaniu systemów informatycznych są powszechnie znane. W metodologii AOP przecinające się zagadnienia są modularyzowane poprzez identyfikacje jasnych ról dla każdego z nich w naszym systemie grid. Implementacja każdego zagadnienia w jego własnym module i luźne powiązania (*loosely coupling*) między modułami mogą znacznie ograniczyć liczbę modułów.

Kolejną zaletą wprowadzenia rozwiązań AOP jest łatwiejszy sposób testowania systemu. Wyizolowane moduły (zagadnienia) możemy łatwiej testować, stosując technikę testów jednostkowych[5] – *Mock object*[6]. Obiekt Mock jest makietą prawdziwego obiektu, którego zachowanie można kontrolować w czasie testów. Obiekty Mock są tworzone w celu przetestowania innych obiektów systemu. Przypomina to testy zderzeniowe samochodów z manekinami, aby zbadać zachowanie się ludzkiego ciała w wypadku.

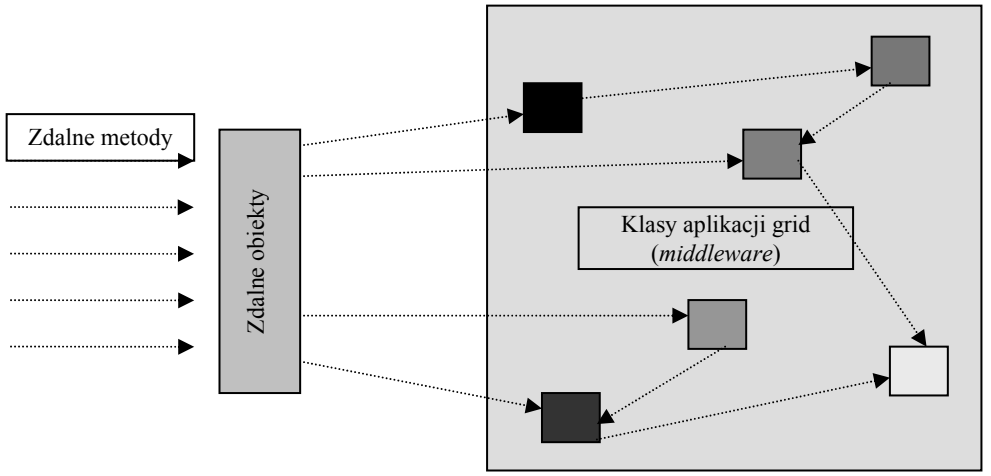
[3] http://en.wikipedia.org/wiki/Grid_computing.

[4] http://en.wikipedia.org/wiki/Design_pattern_%28computer_science%29.

[5] http://en.wikipedia.org/wiki/Unit_testing.

[6] http://en.wikipedia.org/wiki/Mock_object.

Poniżej przedstawiono ogólny szkic systemu grid stworzonego bez podejścia AOP (rys. 1) oraz propozycję systemu stworzonego z udziałem AOP (rys. 2). W pierwszym przypadku zauważamy, że wszystkie klasy systemu znajdują się w jednej „przestrzeni" systemu. W przypadku małych rozwiązań jest to jak najbardziej korzystne, jednak gdy liczba klas idzie w setki, pojawia się wiele problemów, jak np.: spójność systemu, redundancja kodu, utrudnione zarządzanie projektem, problematyczne testy jednostkowe danej funkcjonalności czy też trudniejsze identyfikowanie błędów i ich naprawa.



Rys. 1. Standardowy model aplikacji typu grid

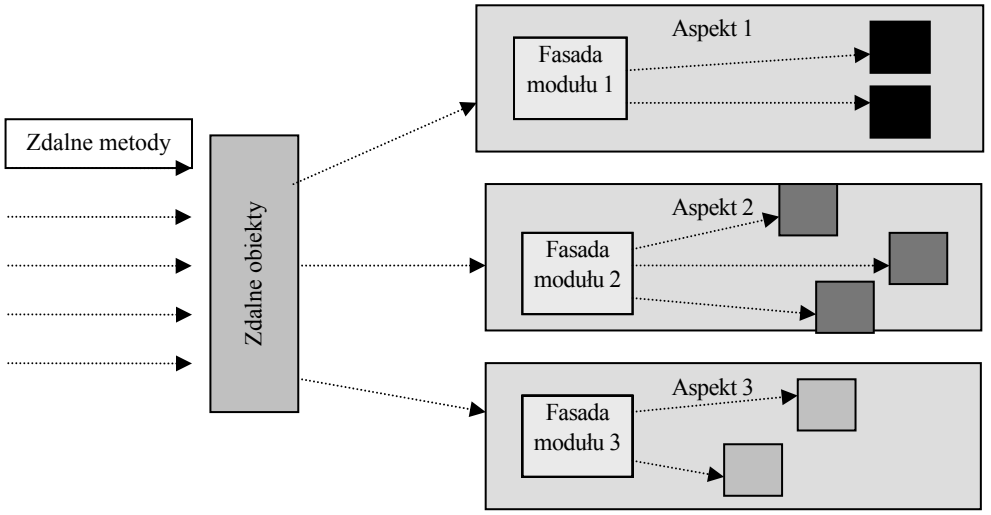
Źródło: opracowanie własne.

Propozycja systemu stworzonego z udziałem AOP (rys. 2) łączy wspólne przecinające się zagadnienia w dany aspekt czy też moduł. Modułem może być np. monitoring systemu grid, rejestracja błędów, zdarzeń czy też autoryzacja użytkowników. Standardem jest budowanie systemów grid z użyciem wielowątkowości. Jak najbardziej możemy wydzielać wspólne zagadnienia z poszczególnych wątków i łączyć je w moduły odpowiedzialne za daną funkcjonalność. W przypadku systemów wielowątkowych musimy, niezależnie od tego, czy stosujemy AOP, czy też nie, zastosować synchronizację wątków. W przypadku rozwiązania AOP możemy zauważyć wprowadzenie wzorca projektowego Fasada[7]. W proponowanym rozwiązaniu doskonale nadaje się on do zbudowania interfejsu danego aspektu warstwy *middleware* gridu.

W chwili obecnej główne wysiłki twórców technologii *grid computing* skupiają się na stworzeniu maksymalnie przezroczystego systemu i zapewnieniu odpowied-

[7] http://en.wikipedia.org/wiki/Facade_pattern.

nio wysokiego poziomu jakości usług jego użytkownikom. W momencie bowiem, gdy użytkownik przedkłada zadanie systemowi gridowemu, do akcji wkracza cały szereg wyspecjalizowanych modułów; moduł odpowiedzialny za interpretacje zapytania użytkownika dokonuje jego analizy i odwołuje się do kolejnego modułu odpowiedzialnego za wyszukiwanie zasobów. Ten z kolei, na podstawie wiedzy o zasobach i poprzednich wykonaniach zadań, stara się wyszukać zasoby spełniające podstawowe kryteria zadanego przetwarzania.



Rys. 2. Model aplikacji typu grid z wykorzystaniem podejścia aspektowego

Źródło: opracowanie własne.

Przedstawiony powyżej podstawowy szkielet budowy oprogramowania gridowego wydaje się być bardzo prosty, jednak jego praktyczna realizacja nastręcza wiele problemów. Należą do nich m.in.: opracowanie sprawnie komunikujących się modułów, opracowanie metody podziału zasobów uwzględniających szeregowanie zadań poszczególnych użytkowników czy wreszcie zagadnienie równoważenia obciążenia zasobów. Autor uważa, że stosowanie architektury aspektowej w tworzeniu systemów grid może znacznie ograniczyć powyższe problemy.

5. Podsumowanie

Możemy stwierdzić, że programowanie aspektowe jest przydatnym mechanizmem w tworzeniu systemów typu grid. Oprócz pomocy w procesie projektowania i poprawy jakości testów, programowanie aspektowe skupia się na separacji zagadnień, zwłaszcza zagadnień przecinających się, co jest bardzo ważne w „oszczędzaniu”

kodu. Modułowa implementacja zagadnień skutkuje systemem łatwiejszym w zrozumieniu i łatwiejszym w utrzymaniu. Aspektowość znacznie pomaga w ulepszaniu testowania systemów rozproszonych (w tym grid). Ponadto możemy zaobserwować, że użyty wzorzec projektowy (Fasada) może być łatwo zastosowany do innych zagadnień, które pojawią się w przyszłości. Zastosowanie AOP w przypadku systemów typu grid wciąż wymaga zbadania. Autor uważa, że zastosowanie AOP pomoże zaimplementować takie przecinające się zagadnienia, jak: monitoring systemu grid, wykrywanie błędów, logowanie, autoryzacja. Implementacja tych zagadnień jest trudna do modularyzacji przy użyciu wyłącznie programowania obiektowego (*Object Oriented Programming*). Jeżeli nie zaczniemy modularyzacji na wczesnym etapie, to doprowadzi nas to do problemów z utrzymanie kodu (projektu) i jego przejrzystością. Autor uważa, że stosując rozwiązania architektoniczne AOP w tworzeniu systemów grid, uczynimy je łatwiejszymi w budowie i utrzymaniu oraz bardziej przejrzystymi. Otwartą kwestią jest, jak bardzo AOP może usprawnić budowanie systemów typu grid.

Literatura

Colyer A., Andy Clement A., Harley G., Webster M., *Eclipse AspectJ*: *Aspect-Oriented Programming with AspectJ and the Eclipse AspectJ Development Tools*, Addison Wesley Professional, 2004.

http://misto.e-informatyka.pl/papers/aop-thesis.pdf Stochmiałek M.: *Programowanie aspektowe: studium empiryczne*.

http://portal.ics.agh.edu.pl:8001/papers/TR-01-1.pdf Słowikowski P.: *Programowanie aspektowe*.

http://wazniak.mimuw.edu.pl/images/e/ea/Zpo-12-wyk.pdf Walter B.: *Zawansowane projektowanie obiektowe – programowanie aspektowe*.

http://www.aosd.net/2007/program/industry/I6-AspectDesignPrinciples.pdf ***Wampler D.:*** Aspect-Oriented Design Principles: Lessons from Object-Oriented Design.

http://www.cs.ubc.ca/~gregor/papers/kiczales-ECOOP2001-AspectJ.pdf Kiczales G. AspectJ Overview.

https://home.agh.edu.pl/~olekb/wyklady/aop.pdf Bieniasz S.: *Programowanie aspektowe AOP*.

Jacob B., *Introduction to Grid Computing*, IBM Corp., 2005.

Jacobson I., Ng P.W., *Aspect-Oriented Software Development with Use Cases*, Addison Wesley Professional, 2004.

Maciaszek L.A., Liong B.L., *Practical Software Engineering*, Addison-Wesley Professional, 2004.

Miles R., *AspectJ Cookbook*, O'Reilly, 2005.

Owoc M.L., Hauke K., *Grid Computing versus Cloud Computing Concepts from User Perspectives*, Advanced Information Technologies for Management AITM'2009, Wrocław 2009.

Owoc M.L., *Przetwarzanie siatkowe – infrastruktura i własności istotne dla współczesnych przedsiębiorstw*, Prace Naukowe Akademii Ekonomicznej nr 1044, AE, Wrocław 2004.

Owoc M.L., *Research Trends in Knowledge Grid*, Research Papers, red. A. Nowicki, University of Economics, Wroclaw 2009.

Owoc M.L., Sachańbiński J., *Wybrane funkcje administrowania zasobami informatycznymi w przetwarzaniu siatkowym*, Prace Naukowe Akademii Ekonomicznej nr 1122, red. E. Niedzielska, H. Dudycz, M. Dyczkowski, AE, Wrocław 2005.

Owoc M.L., Walasiński T., *Przetwarzanie siatkowe jako przełom w technologiach informatycznych*, Studia Informatica'2006, Uniwersytet Szczeciński, Szczecin 2006.

Stockinger H., *Defining the grid: a snapshot on the current view*, Springer Science+Business Media, 2007.

AOP ARCHITECTURAL SOLUTIONS IN THE CONSTRUCTION OF GRID SYSTEMS

Summary: In currently being built computer systems grid processing is becoming increasingly important. The methods of construction and the architecture of grid system architecture constantly evolve. All the time we search for universal solutions to enhance the reliability, clarity and speed of these systems. One of the ideas is to introduce a solution based on the aspect of AOP programming. Grid system, which is a variant of dispersed system, accomplishes functionality in some of its nodes. However, according to the most popular way to create software so far (OOP, Object Oriented Programming) some recurring functionalities are implemented in different nodes. Architecture and programming solutions of AOP enable to isolate these functionalities and close them in the aspect and then implement them in one place. This article is an attempt to show possible applications of AOP in grid systems and to demonstrate the advantages of doing so. This article explains the pre-conceptions about AOP programming grid-type systems and tries to find the common ground for these technologies.