Krzysztof ZIELIŃSKI
AKADEMIA GÓRNICZO-HUTNICZA, WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I ELEKTRONIKI,
KATEDRA INFORMATYKI

Adaptowalne i reflektywne warstwy pośredniczące jako podstawa autonomicznych systemów komputerowych

Prof. dr hab. inż. Krzysztof ZIELIŃSKI

Prof. dr hab. inż. Krzysztof Zieliński jest kierownikiem Katedry Informatyki AGH na Wydziale EAIiE. Jego zainteresowania naukowe dotyczą rozproszonych systemów komputerowych, sieci komputerowych i architektur oprogramowania tych systemów ze szczególnym uwzględnieniem architektur udostępniania serwisów sieciowych oraz adaptowalnych systemów oprogramowania wykorzystujących informacje semantyczną. Bezpośrednio kieruje Grupą Systemów Rozproszonych. Jest autorem ponad 150 publikacji naukowych.

e-mail: kz@agh.edu.pl

Streszczenie

Artykuł omawia problematykę prac naukowo-badawczych dotyczących adaptowalnych reflektywnych systemów warstw pośredniczących. Prace te koncentrują się na mechanizmach zarządzania zwirtualizowanymi zasobami komputerowymi oraz ich monitorowaniu. Przedstawione rozważania zilustrowano kilkoma przykładami systemów zrealizowanych w Katedrze Informatyki AGH. Prace te dotyczą między innymi monitorowania infrastruktury systemów typu Grid oraz zasobów systemu Solaria 10.

Słowa kluczowe: warstwy pośredniczące, Gridy, wirtualizacja, systemy reflektywne, systemy adaptowalne.

Adaptable and Reflective Middleware as Foundation for Autonomic Computing

Abstract

The paper describes research activity related to the adaptable reflective middleware. These studies focus on virtualization mechanisms of computer resources and their monitoring. The presented considerations are illustrated by a few examples of the systems implemented at the Department of Computer Science at AGH UST. These systems concern monitoring and adaptive management of Grid systems infrastructure and Solaris 10 virtualized resources, among others.

Keywords: middleware, Grids, virtualization, reflective systems, adaptable systems.

1. Wprowadzenie

Zarządzanie zasobami systemów rozproszonych z wykorzystaniem warstw pośredniczących stanowi przedmiot badań prowadzonych od kilku lat w Grupie Systemów Rozproszonych w Katedrze Informatyki AGH. Badania te wpisują się w nurt prac dotyczących systemów Gridowych i są realizowane w ramach projektów europejskich, grantów Ministerstwa na Nauki i Szkolnictwa Wyższego oraz we współpracy z przemysłem.

Zarządzanie zasobami rozproszonego systemu komputerowego, będące przedmiotem tych prac, jest rozumiane jako dążenie do efektywnego podziału infrastruktury obliczeniowej i komunikacyjnej pomiędzy wiele równocześnie przetwarzanych aplikacji w taki sposób, aby zrealizować odpowiednią strategie działania systemu oraz zapewnić wymaganą jakość usług świadczonych na rzecz użytkownika systemu.

Tematyka realizowanych prac należy do nurtu intensywnych badań prowadzonych przez wiodące firmy takie jak IBM, Sun Microsystems, HP, Oracle czy Veritas w zakresie inteligentnego dopasowywania wymagań aplikacji do aktualnych możliwości systemu. Zagadnieniom tym poświęcone jest szereg projektów badawczych; w ich nazwach [26] występują określenia takie jak: *utility computing*, *on-demand computing*, *grid computing*, *service-based computing*, *adaptive management*, *autonomic computing* [19], a nawet *organic computing* [10]. Chociaż projekty te różnią się pewnymi aspektami, wpisują się one dobrze w zakres adaptowalnych systemów rozproszonych klasy *middleware* bazując na kilku kluczowych koncepcjach, do których należą:

1. wirtualizacja zasobów obliczeniowych, pamięciowych i komunikacyjnych,
2. wprowadzenie mechanizmów dynamicznego zarządzania konfiguracją złożonego systemu, umożliwiających udostępnianie zasobów na żądanie oraz realizację autonomicznej strategii działania,
3. strukturalizacja zasobów systemu w formie kontenerów monitorowanych i niezależnie zarządzanych.

Koncepcje te stwarzają podstawy systemowe dla budowy adaptowalnych i reflektywnych warstw pośredniczących (*middleware*).

Podstawę prowadzanych badań stanowi przekonanie [29], że nowej generacji systemy Gridowe muszą być konstruowane na bazie warstw pośredniczących wykorzystujących możliwości systemowe, jakie stwarzają nowoczesne systemy operacyjne, a procesy adaptowalności i refleksji są kluczowe dla osiągnięcia pożądanego celu działania systemu. Podejście to w znaczący sposób uzupełnia prace nad systemami Gridowymi, które koncentrują się zasadniczo na bardzo wysokim poziomie abstrakcji i dotyczą budowy usług systemowych, specyfikowanych jako usługi Webowe.

Niniejszy artykuł zawiera przegląd aktualnie prowadzonych prac w Katedrze Informatyki AGH w zakresie systemów adaptowalnych i reflektywnych. Prace te wykorzystują najnowocześniejsze systemy operacyjne i platformy sprzętowe i opierają się na zaawansowanych technikach wirtualizacji zasobów na różnych poziomach abstrakcji. Na ich bazie budowane są odpowiednie elementy warstwy pośredniczącej, które mogą być wykorzystywane w procesie zarządzania zasobami z poziomu aplikacji lub dedykowanych systemów zarządzania. W ten sposób mogą one stanowić bazę dla konstrukcji autonomicznych systemów komputerowych.

Struktura niniejszego artykułu jest następująca. W punkcie 2 omówiono podstawowe pojęcia dotyczące adaptowalnych reflektywnych warstw pośredniczących. Następnie w punkcie 3 przedstawiono zagadnienia wirtualizacji systemów komputerowych. W kolejnych dwu punktach skoncentrowano się odpowiednio na zagadnieniu monitorowania zasobów systemu komputerowego oraz zarządzaniu zasobami. W punkcie 6 przedstawiono wybrane wyniki prac eksperymentalnych. Artykuł kończy krótkie podsumowanie i spis literatury.

2. Podstawy adaptowalnych i reflektywnych warstw pośredniczących

Oprogramowanie warstwy pośredniczącej było tradycyjnie projektowane dla stosunkowo zwartych systemów rozproszonych. W ostatnim okresie pojawiły się nowe wymagania odnośnie tych platform związane z potrzebami ogromnej skalowalności wynikającej z wszechobecności (ang. *ubiquitous and pervasive computing*) i mobilności systemów przetwarzania (ang. *mobility computing*) w bardzo zróżnicowanym środowisku z punktu widzenia dostępnych zasobów obliczeniowych, pamięciowych i komunikacyjnych. Sprawia to, że platformy te muszą posiadać możliwość zbierania informacji o stanie otoczenia i dostosowywania się do zmieniających się warunków pracy.

Adaptacyjne reflektywne techniki są postrzegane za podstawę rozwoju następnej generacji platform *middleware* [11, 22]. Umożliwiają one, bowiem ich adaptację, czyli dostrojenie w celu zaspokojenia wymagań środowiska bądź aplikacji. Adaptacja ta może

przebiegać autonomicznie lub półautonomicznie na podstawie parametrów uruchomienia (ang. *deployment*) aplikacji w ramach zdefiniowanej polityki działania użytkowników i administratora [4].

Z konstrukcją systemów adaptacyjnych związanych jest pięć podstawowych pojęć [22]: pomiar, raportowanie, sterowanie, sprzężenie zwrotne oraz stabilność, których znaczenie w odniesieniu do platform *middleware* zdefiniowano w pracach [3, 6]. Koncepcję reflektywnego programowania (ang. *reflective programming*) opracował natomiast Brian Smith w MIT [1, 23]. W odniesieniu do platform klasy *middleware*, reflektywność oznacza dostępność mechanizmów do wewnętrznej manipulacji serwisami systemu w trakcie ich działania. Reflektywne systemy *middleware* stanowią zatem właściwą podstawę do budowy adaptowalnych i elastycznych aplikacji rozproszonych [15]. Mechanizmy reflektywne są konieczne do wyzwolenia działania mechanizmów adaptowalności.

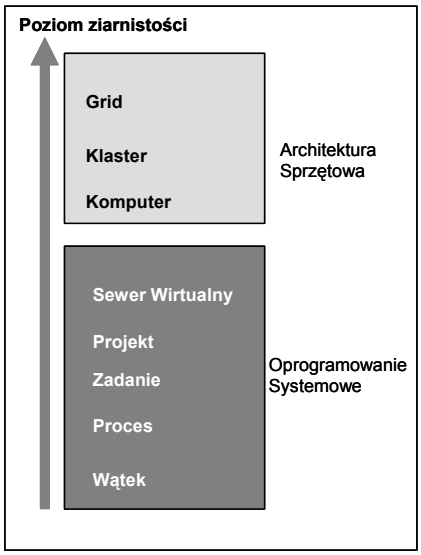
Prace w zakresie mechanizmów refleksji wiążą się ściśle z zagadnieniami metaprogramowania i użycia odpowiednich protokołów MOP (*Meta-Object Protocol*) wprowadzonych po raz pierwszy przez Gregora Kiczales [14]. Istota tego podejścia zasadza się na wyróżnieniu w systemie dwu poziomów: poziomu bazowego, który zapewnia odpowiednią funkcjonalność i poziomu meta, zawierającego polityki działania i zachowania się systemu. Zmiany na poziomie meta pozwalają na modyfikację działania systemu bazowego. System bazowy musi w tym celu udostępniać metainterfejsy, które są dostępne dla poziomu meta. Projektowanie metainterfejsów oraz MOP jest, zatem działaniem podstawowym w zakresie wykorzystania mechanizmów reflektywnych. Interfejsy te muszą być wystarczająco ogólne, aby umożliwić nieprzewidziane zmiany platformy a jednocześnie dostatecznie restrykcyjne, aby nie spowodować dezintegracji systemu [4].

Rosnąca złożoność systemów informatycznych i dążenie do efektywnej ich implementacji sprawia, że w ostatnim okresie podejście komponentowe do ich budowy staje się dominujące. Komponent jest zazwyczaj definiowany, jako enkapsułowana funkcjonalność, która może być częścią większej aplikacji, posiadająca mechanizmy umożliwiające jej łączenie z innymi komponentami w celu stworzenia systemu o pożądanej funkcjonalności [7, 24]. W tym celu komponenty muszą być wyposażone w odpowiednie interfejsy i implementować odpowiedni protokół interakcji. Kolekcja takich interfejsów oraz protokół interakcji definiuje środowisko komponentowe (ang. *component framework*). Interfejsy te oraz protokół bazują bezpośrednio na usługach i protokole interakcji udostępnianych w ramach platform *middleware* przez kontenery, w których komponenty muszą być osadzone. W ten sposób własności środowisk komponentowych wiążą się bezpośrednio z zagadnieniami budowy nowoczesnych platform *middleware*. Odnosi się to w pełni do takich środowisk komponentowych jak Enterprise Java Beans (EJB) [7], oraz CORBA Component Model (CCM) [24, 25].

Realizowane prace wykorzystują także pewne elementy związane z badaniami w zakresie budowy spontanicznych kontenerów (ang. *spontaneous containers*) [9, 21]. Kontenery stanowią środowisko wykonania komponentów, a ich główne zadanie polega na zapewnieniu komponentom dostępności usług, takich jak np. usługi transakcyjne, bezpieczeństwa, komunikacja zdalna itp. Podstawowym celem wprowadzenia modelu kontenerowego jest oddzielenie funkcjonalności użytkowej od usług systemowych. Koncepcja spontanicznych kontenerów polega na zapewnieniu mechanizmów dynamicznej zmiany udostępnianych przez kontener usług.

3. Wirtualizacja zasobów komputerowych

Podstawową technikę udostępniania i zarządzania zasobami komputerowymi stanowi ich wirtualizacja. Polega ona na przykryciu fizycznych zasobów odpowiednim oprogramowaniem, którego interfejs (API) stanowi ekspozycję ich funkcjonalności przydatną z punktu widzenia określonego celu wykorzystania tych zasobów. Współczesne systemy komputerowe charakteryzują się rosnącą liczbą poziomów wirtualizacji, które schematycznie, w układzie rosnącej ziarnistości, przedstawiono na rysunku 1.



Rys. 1. Poziomy wirtualizacji w systemach komputerowych
Fig. 1. Virtualization layers of computer systems

Obecnie obserwuje się tendencje do implementacji najniższych poziomów wirtualizacji programowej w formie dedykowanych podsystemów procesorów. Równocześnie nowoczesne systemy operacyjne cechują się zwiększeniem liczny poziomów wirtualizacji. Bardzo dobrym przykładem jest tutaj system Solaris 10, który wyróżnia wszystkie 5 poziomów wirtualizacji systemowej pokazane na rysunku 1. Oprócz wątków i procesów, typowo występujących w każdym systemie, wprowadzono w systemie Solaris 10: zadania stanowiące grupę procesów, projekty będące kontenerami procesów, którym można przypisywać zasoby dzielone przez procesy, oraz wirtualne serwery, zwane strefami, które rozszerzają koncepcje projektów o pełną izolacje przebiegu obliczeń w poszczególnych strefach.

Wprowadzenie wielopoziomowej wirtualizacji nie oznacza, że poszczególne poziomy są udostępnione w jednolity sposób dla twórców oprogramowania aplikacyjnego skonstruowanym w danym środowisku programowania. Monitorowanie zwirtualizowanych zasobów oraz zarządzanie nimi z języka poziomu języka Java wymaga dodatkowej warstwy oprogramowania pośredniczącego, którego budowa była przedmiotem prac prowadzonych w Katedrze Informatyki AGH. Prace te dotyczyły wszystkich warstw wirtualizacji od poziomu procesu, aż po architektury typu Grid [20].

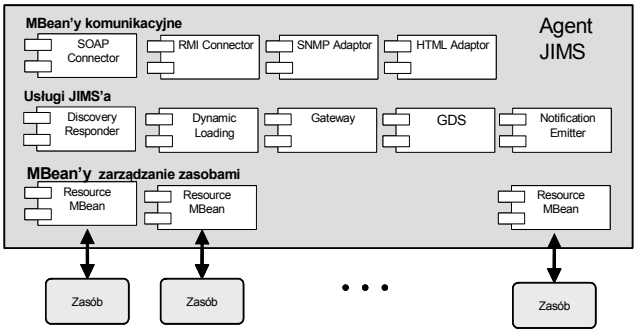
4. Monitorowanie zasobów

Monitorowanie zasobów rozproszonych systemów komputerowych jest przedmiotem prac prowadzonych w Grupie Systemów Rozproszonych w Katedrze Informatyki AGH od kilku lat. W wyniku tych prac, prowadzonych w projekcie CrossGRID oraz Clusterix powstał system monitorowania JIMS [2, 18, 28] (JMX-based Infrastructure Monitoring System). Zgodnie z nazwą system ten wykorzystuje rozszerzenie języka Java i wyprowadzające zarządzane obiekty (MBean), które mogą reprezentować zasoby komputerowe. Wybór technologii JMX okazał się bardzo trafny, gdyż obecnie stanowi ona standardową część środowiska języka Java i coraz więcej aplikacji posiada wbudowane możliwości zarządzania z użyciem tej technologii..

Architektura systemu JIMS została przedstawiona schematycznie na rysunku 2. Składa się ona z trzech podstawowych warstw:

1. Instrumentacji zasobów – każdy zasób jest reprezentowany przez MBean'a, który udostępnia jego parametry,

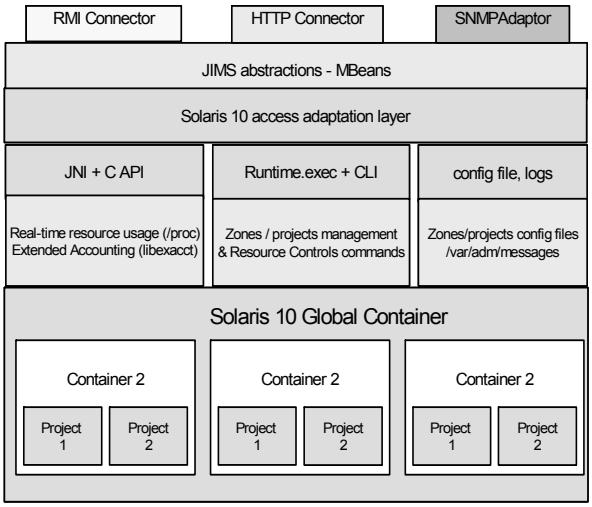
2. Usług systemu JIMS – usługi te umożliwiają: wyszukiwanie zarządzanych zasobów, oraz bram (Gateway), dynamiczne ładowanie Mbeanów monitorujących oraz bram, globalny system wyszukiwania bram (Global Discovery Gateway) oraz system dystrybucji zdarzeń,
3. Komunikacji – warstwa ta składa się z konektorów: RMI, SOAP, SNMP oraz i adapterów takich jak HTML i SNMP.



Rys. 2. Architektura systemu JIMS
Fig. 2. JIMS system architecture

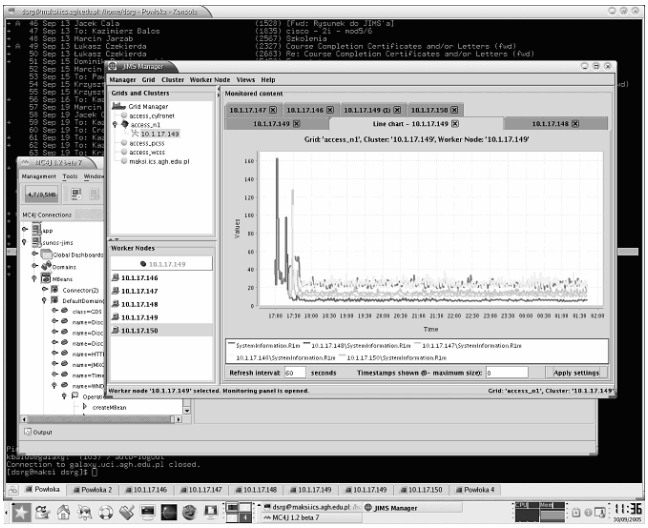
Architektura JIMS'a zadecydowała o jego uniwersalności i szerokim spektrum zastosowań. Jest to system posiadający, dzięki usługom wyszukiwania instrumentowanych zasobów, własność autokofiguracji zarówno w sieci lokalnej, jak i sieci zdalnej, oraz łatwość współpracy z praktycznie wszystkimi podstawowymi technologiami budowy systemów rozproszonych. Na szczególną uwagę zasługuje możliwość udostępnienia funkcjonalności JIMS'a przez Web Serwisy.

Wymienione cechy systemu JIMS zadecydowały o jego wyborze w celu rozbudowy o moduły monitorowania i zarządzania zwirtualizowanymi zasobami komputerowymi udostępnianymi przez system Solaris 10. Wynikiem tych prac [28] jest system pozwalający na zarządzanie z poziomu języka Java zdaniami, projektami i strefami systemu Solaris 10, jak również zbierania danych z systemu Extended Accounting. Jest to system bardzo złożony ze względu na rozbudowaną warstwę instrumentacji, zasobów systemowych. Idę jego budowy prezentuje rysunek 3.



Rys. 3. Architektura rozszerzenia sytemu JIMS dla systemu Solaris 10
Fig. 3. Extension of JIMS architecture for Solaria 10

Rozszerzenia systemu JIMS o możliwości zarządzania zasobami systemu Solaris 10 umożliwiły podjęcie interesujących prac eksperymentalnych dotyczących adaptowalnego zarządzania zasobami tego systemu. Rozszerzają one istotnie obszar użycia systemu JIMS, który z powodzeniem był eksploatowany do monitorowania europejskiego Gridu w projekcie CrossGrid oraz Gridu krajowego w projekcie Clusterix. Na rysunku 4 przestawiono przykład interfejsu graficznego systemu JIMS umożliwiającego zbieranie charakterystyki obciążenia farmy procesorów. JIMS umożliwia monitorowanie systemów pracujących pod różnymi systemami operacyjnymi, a jego wyjściowa wersja obejmuje system Linux oraz Windows.



Rys. 4. Interfejs graficzny systemu JIMS
Fig. 4. JIMS graphic interface

5. Zarządzanie zasobami

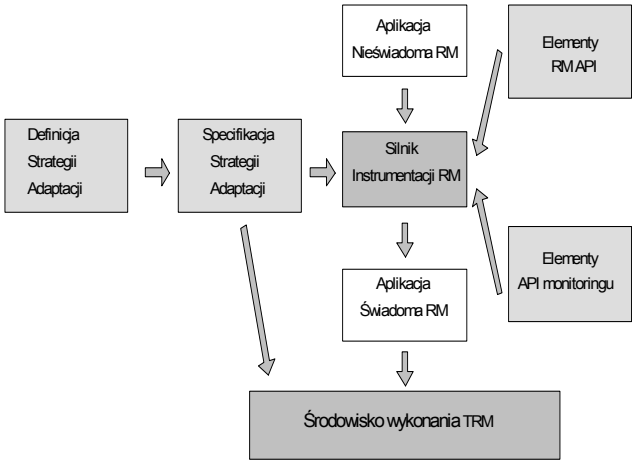
Zarządzanie zasobami komputerowymi ma na celu dynamiczny ich podział pomiędzy aplikacje obliczeniowe tak, aby zagwarantować założone warunki ich wykonania. Potrzeba taka istnieje w wielu systemach komputerowych, a w szczególności interaktywnych systemach typu Grid opartych na usługach. Prace zakresie zarządzania zasobami prowadzone w Katedrze Informatyki AGH nawiązują do projektu MVM [17] firmy SUN Microsystems w ramach, którego został zaproponowany interfejs do zarządzania zasobami wielozadaniowej maszyny wirtualnej Java MVM. Na bazie tego interfejsu została zaproponowana nowa bardziej ogólna specyfikacja standardu zarządzania zasobami JSR 284 (*Resource Consumption Management API* – RM API) [12].

Zrealizowane prace dotyczą trzech ściśle powiązanych obszarów:
1. Zarządzania zasobami pojedynczego procesu reprezentującego maszynę wirtualną MVM lub zasobami klastera takich maszyn.
2. Wykorzystania koncepcji zawartych w JSR 284 do zarządzania zasobami systemu Solaris 10.
3. Użycia technologii *Policy Management Autnomic Computing* (PMAC) [19] do budowy systemów zarządzania zasobami.

W pierwszym obszarze sformułowano koncepcje TRM (*Transparent Resource Management*) przeźroczystego zarządzania zasobami. Jej podstawę stanowi stwierdzenie, że współczesne aplikacje są na tyle złożone, że trudno wymagać od użytkownika, aby jeszcze zajmował się zagadaniem zarządzania zasobami. Funkcjonalność ta powinna być automatycznie uzupełniona przez środowisko wykonania aplikacji, w taki sposób, aby system wynikowy realizował wybraną przez użytkownika politykę zarządzania zasobami. Elementy składowe środowiska TRM zostały przedstawione schematycznie na rysunku 5.

Punktem centralnym tego środowiska jest silnik instrumentacji, który zgodnie wybraną przez użytkownika polityką instrumentuje aplikacje wyjściową odpowiednimi wywołaniami RM API. W prowadzonych pracach [17] wybrano aspektową (AOP) technikę instrumentacji, a dla specyfikacji polityki adaptacji zastosowano maszynę stanową FSM. W ten sposób rozdzielono problem definiowania strategii adaptacji od jej implementacji oraz kodu aplikacji użytkownika osiągając pożądaną własność przeźroczy-

stości zarządzania zasobami. Wstępne eksperymenty praktyczne w pełni potwierdziły skuteczność tego podejścia.



Rys. 5. Środowisko przetwarzania TRM
Fig. 5. TRM processing environment

Drugi obszar prac dotyczy zagadnienia zastosowania RM API do zarządzania zwirtualizowanymi zasobami systemu Solaris 10. W tym celu zbudowano implementacje RM API w technologii JMX. Pozwoliło to na zastosowanie RM API w środowisku rozproszonym oraz zweryfikowanie jego założeń projektowych.

Trzeci obszar prac związany jest zastosowaniem technologii PMAC, opracowanej przez firmę IBM, do zarządzania zasobami. Prace prowadzone w tym zakresie zmierzają do automatycznej instrumentacji zwirtualizowanych zasobów tak, aby mogły one być zarządzane przez moduły systemu PMAC oraz do rozbudowy tego systemu o elementy dynamicznej konfiguracji. W ten sposób prowadzone prace nawiązują do nurtu prac związanego obliczeniami autonomicznymi.

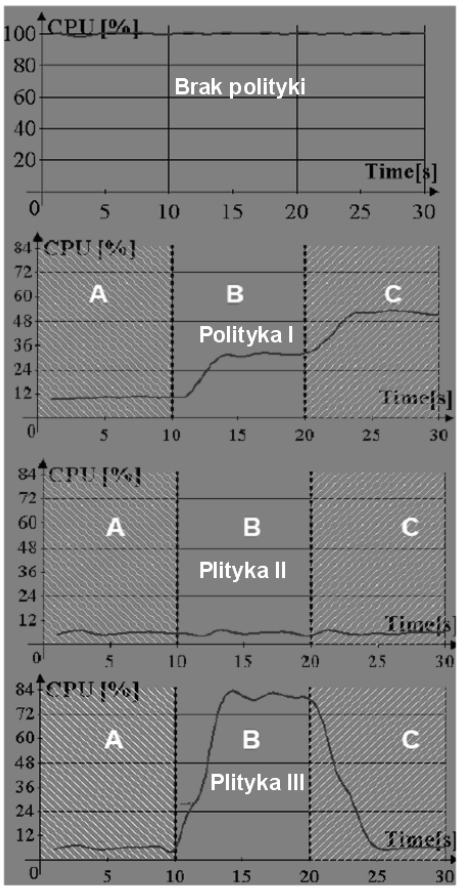
6. Przykłady

Opisane koncepcje zostały zweryfikowane praktycznie. W tym punkcie zostaną przedstawione tylko dwa przykłady ilustrujące skuteczność prezentowanych wcześniej rozwiązań. Szerszy opis wyników prowadzonych prac można znaleźć w [17, 28].

Pierwszy przykład dotyczy badania maszyny MVM w ramach, której wykonuje się aplikacja zmodyfikowana w środowisku TRM. Na rysunku 6 przedstawiono zależność obciążenia procesora od przyjętej polityki. W ramach aplikacji, co 1 [s] uruchamiany jest wątek. Każdy wątek może skonsumować 100% czasu procesora. Jednakże aplikacja posiada wbudowaną politykę określającą dopuszczalny przydział czasu procesora w zależności od liczby wątków. Liczba aktywnych wątków określa, zatem stan aplikacji zgodnie z Tabelą 1. Natomiast Tabela 2 definiuje jedną z trzech badanych polityk. Należy podkreślić, że każda z tych polityka zależy od wyróżnionych stanów aplikacji. Uzyskane rezultaty wskazują na poprawność działania systemu i osiągnięcie zamierzonego celu zarządzania czasem procesora.

Drugi przykład dotyczy przydziału czasu procesora do projektów w systemie Solaris 10. W tym przykładzie wykorzystano rozszerzenia systemu JIMS dla Solaris 10 o możliwości zarządzania projektami oraz implementację RM API w technologii JMX.

Na rysunku 7 pokazano zużycie procentowe czasu procesora odpowiednio przez projekt P1 i P2. Przyjęta polityka przydziału czasu zakłada, że przydział ten zmienia się okresowo w stosunku 80/20 i 20/80. Obrazuje to fala prostokątna pokazana na rysunku 7. Odpowiednio oscyluje przydział czasu do projektów. Potwierdza to możliwość przejęcia pełnej kontroli programowej nad rozważanym zasobem, jakim jest w tym przykładzie przydział procentowy mocy procesora do projektu. Pozwala to na dynamiczne ustalanie zasobów projektu w zależności od potrzeb aplikacji. Szersze omówienie przedstawionych rezultatów można znaleźć w pracach [12].



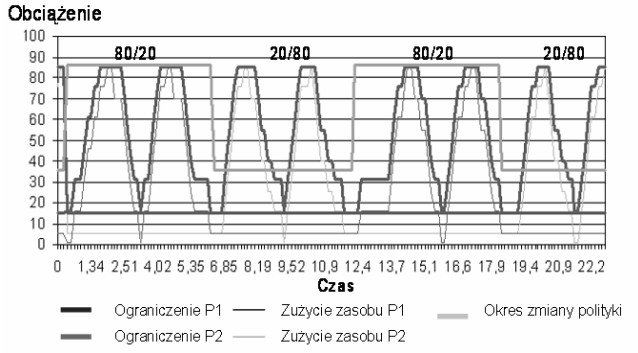
Rys. 6. Zależność obciążenia procesora od polityki
Fig. 6. CPU load vs. policy

Tab. 1. Stany aplikacji
Tab. 1. Policy states

Stan	Liczba wątków W
A	$10 \leq W$
B	$10 < W \leq 20$
C	$20 <$

Tab. 2. Polityki przydziału procesora
Tab. 2. CPU allocation policies

Polityka	Stan		
	A	B	C
I	10%	30%	50%
II	5%	5%	5%
III	5%	80%	5%



Rys. 7. Zależność przydziału CPU do projektu P1 i P2 w zależności od polityki
Fig. 7. CPU allocation for Project P1 and P2 vs. policy

7. Wnioski

Przedstawiony zakres realizowanych prac odnośnie systemów zarządzania zasobami rozproszonego systemu komputerowego stanowi etap prowadzący do skonstruowania systemu autonomicznego, którego funkcjonowanie będzie określone poprzez specyfikacje polityki jego działania. Skonstruowane rozwiązania pozwalają na monitorowanie i zarządzanie zwirtualizowanymi zasobami systemowymi z poziomu języka Java. Opracowano także technikę automatycznej instrumentacji kodu aplikacji tak, aby mogły one wykorzystywać dostępne mechanizmy zarządzania zasobami. Zaproponowane rozwiązania stanowią rozwinięcie opracowywanych standardów i weryfikują ich kompletność. Realizacja tych prac zakłada wykorzystanie mechanizmów systemowych, co podnosi stopień złożoności rozwiązań, lecz jednocześnie prowadzi to do zwiększenia ich efektywności.

Urealnienie wizji budowy autonomicznych systemów komputerowych wymaga dalszych prac, które obecnie koncentrują się na zagadnieniach budowy modeli informacyjnych zasobów, ich opisu semantycznego, oraz ekspozycji w formie Web Serwisów. Prace te zmierzają także to opracowania metod automatyzacji generacji warstwy integrującej zarządzanie zasobami z warstwą zasobów zgodnie z wymaganiami aplikacji.

8. Literatura

[1] Andersen A., Blair G.S., Stabell-Kulo T., Myrvang P.H., Rost T.-A.N. (2003): Reflective Middleware and Security: OOPP meets Obol. Proceedings of the Workshop on Reflective Middleware, Middleware 2003, Rio de Janeiro, Brazylia; Springer-Verlag, Heidelberg, Niemcy

[2] Balos K., Zieliński K.: JIMS- the Uniform Approach to Grid Infrastructure and Application Monitoring, CGW-2004.

[3] Bergmans L., Aksit M. (2000): Aspects and Crosscutting in Layered Middleware Systems. Proceedings of the IFIP/ACM (Middleware 2000) Workshop on Reflective Middleware, Palisades, Nowy Jork, USA

[4] Blair G.S., Costa F.M., Coulson G., Duran H.A., Parlavantzas N., Delpiano F., Dumant B., Horn F., Stefani J.B. (1998): The Design of a Resource-Aware Reflective Middleware Architecture. Proceedings of the Second International Confer¬ence on Meta-Level Architectures and Reflection (Reflection '99), Springer, St. Malo, Francja, s. 115-134

[5] Blair G.S., Coulson G., Andersen A., Blair L., Clarke M., Costa F., Duran-Limon H., Fitzpatrick T., Johnston L., Moreira R., Parlavantzas N., Saikoski K. (2001): The Design and Implementation of Open ORB 2. IEEE Distributed Systems Online, 2(6)

[6] Corsaro A., Wang N., Venkatasubramanian N., Coulson G., Costa F.M. (2003): The 2nd Workshop on Reflective and Adaptive Middleware. Proceedings of the Middleware 2003, Rio de Janeiro, Brazylia

[7] Coulson G. (2002): What is Reflective Middleware? IEEE Distributed Systems Online, http://dsonline.computer.org/middleware/RMarticlel htm

[8] DeMichiel L.G. (2002): Enterprise JavaBeansTM Specification, Version 2.1. 2002, Sun Microsystems, Inc.

[9] Favarim F., Siqueira F., Fraga J. (2003): Adaptive Fault-Tolerant CORBA Components. Proceedings of the 2nd Workshop on Reflective and Adaptive Middleware, Middleware 2003, Rio de Janeiro, Brazylia

[10] Frei A., Popovici A., Alonso G. (2004): Eventizing Applications in an Adaptive Middleware Platform. Technical Report 451, Department of Computer Science, ETH Zürich, Szwajcaria

[11] Ganek A., Corbi T. (2003): The Dawning of the Autonomic Computing Era. IBM Systems Journal, 42(1), s. 5-18

[12] Gawęda I., Wieczorek D. (2006): An environment for adaptive management of Solaris 10 using JSR-284 Resource Consumption Management API, M.Sc.thesis DCS AGH-UST

[13] Geihs K. (2001): Middleware Challenges Ahead. IEEE Computer, 34(6), 24-31.

[14] Kon F., Blair G.S., Campbell R.H. (2000): Workshop on Reflective Middleware. Proceedings of the IFIP/ACM Middleware 2000, Nowy Jork, USA

[15] Kon F., Costa F., Blair G., Campbell R.H. (2002): The Case for Reflective Middleware. Communications of the ACM, 45(6), 33-38

[16] Jarząb M., Kosinski J., Zieliński K. (2005): Configuratiom Management of Massively Software Engineering Evolution and Emerging Technologies, Scalable Systems, Frontiers in Artifical Intelligence and Applications, IOS Press

[17] Janik, A., Zieliński K.(2006): Transparent Resource Management with RM API, LNCS Springer ICCS

[18] Midura J., Balos K., Zieliński K.(2004): Global Discovery Service for JMX Architecture, LCNS Springer, ICCS

[19] Murch R. (2004): Autonomic Computing, IBM Press, Prentice Hall

[20] NGG2 Group (2004): Next Generation Grids 2. Requirements and Options for European Grids Research 2005-2010 and Beyond. Expert Group Report

[21] Popovici A., Frei A., Alonso G. (2003): A proactive middleware platform for mobile computing. In: Proc. of the 4th ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazylia

[22] Schantz R.E., Schmidt D.C. (2001): Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications. Encyclopedia of Software Engineering, Nowy Jork, Wiley & Sons s. 801-813

[23] Smith B.C. (1982) Procedural Reflection in Programming Languages, PhD Thesis, MIT Laboratory of Computer Science

[24] Object Management Group (2002): CORBA Components. http://www.omg.org/cgi-bin/doc?formal/02-06-65

[25] Object Management Group: Deployment and Configuration of Component-based Distributed Applications Specification. http://www.omg.org/docs/ptc/03-07-02.pdf

[26] Sun Microsystems (2004): Utility Computing from SUN Microsystems. Sun Microsystems, Inc. Business White Paper

[27] Szydło T., Szymacha R., Zieliński K. (2006) Policy-based Context-aware Adaptable Software Components for Mobility Computing, accepted for IEEE EDOCS Hon-Kong

[28] Zieliński K., Jarząb M., Wieczorek D., Bałos K.(2006) JIMS Extensions for Resource Monitoring and Management of Solaris 10, LNCS Springer

[29] Zieliński K., Jarząb M., Kosinski J.(2005): Role of N1 Technology in the Next Generation Grids Middleware, LCNS Springer

Artykuł recenzowany