Seria: INFORMATYKA z. 30 Nr kol. 1315

Jakub SZYMASZEK
Aleksander LAURENTOWSKI
Krzysztof ZIELIŃSKI

SYSTEM MONITOROWANIA HETEROGENICZNYCH PROGRAMOWO OBIEKTOWYCH ŚRODOWISK ROZPROSZONYCH

Streszczenie. Artykuł omawia prototypowy system monitorowania i wizualizacji aplikacji rozproszonych, zrealizowanych na podstawie modelu współdziałania komponentów uruchomionych w różnych środowiskach rozproszonych. System ten, zwany MODIMOS, jest rozszerzalny, tzn. umożliwia łatwą adaptację wielu środowisk dla swoich potrzeb. System stosuje różne metody zarządzania informacją dla ograniczenia strumienia i złożoności danych wejściowych.

A MONITORING SYSTEM FOR SOFTWARE-HETEROGENEOUS DISTRIBUTED OBJECT-BASED ENVIRONMENTS

Summary. This paper describes a prototype tool, called Managed Object-based Distributed Monitoring System. MODIMOS is a project aimed at development of an adaptable platform for visualization of distributed applications, built of interoperating heterogeneous components. MODIMOS is expandable, allowing to add new monitored environments. It employs various management mechanisms to cope with gathered information size and complexity.

Praca częściowo finansowana w ramach grantu KBN nr 8 S503 015 06.

UN MONITEUR POUR LES ENVIRONS HÉTÉROGÉNIQUES OBJET ORIENTÉS REPARTIS

Résumé. Cet article décrit un système prototypique d'observation et visualisation des applications hétérogéniques objet orienteés repartis, qui sont realiseés par un modèle de coopération des composants qui marchent dans les environements différents. Cet système, appelé MODIMOS (Managed Object-based Distributed Monitoring System) facilite l'adaptation de plusieurs environements. Il adapte les méthodes differentes de gestion d'information, pour limiter le flux et la complexité de donneées.

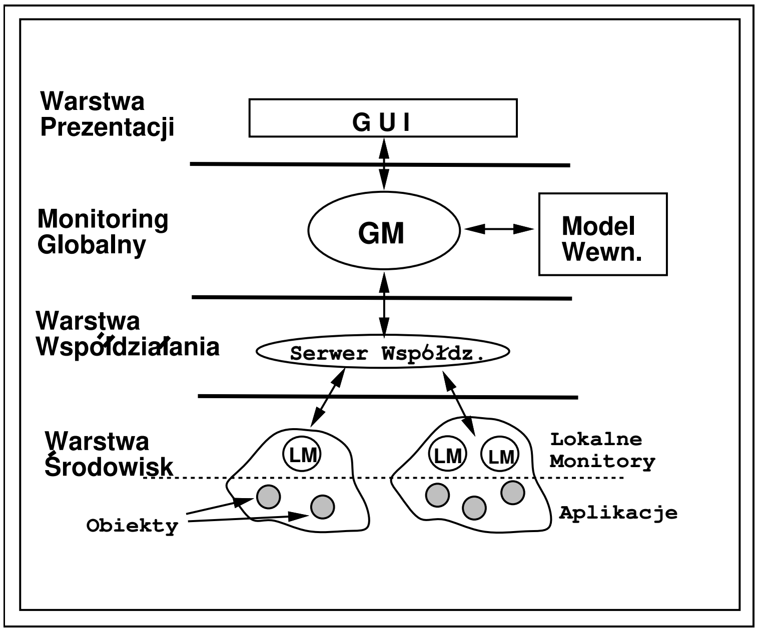
1. Wstęp

Nie istnieją uniwersalne systemy informatyczne i aplikacje rozwiązujące dowolny problem użytkownika w jego środowisku. Każdy problem najlepiej (najszybciej, najprościej) daje się rozwiązać w ramach określonego modelu przetwarzania. Na przykład obliczeń najlepiej dokonywać w językach i modelach imperatywnych (np. w językach FORTRAN, C, C++), podczas gdy moduły decyzyjne najprościej i najbardziej elegancko zdefiniować można w językach deklaratywnych (np. oferowanych przez Lisp czy Prolog). Wśród środowisk programowania rozproszonego w sieciach komputerowych powstało w ostatnich latach wiele systemów reprezentujących różne modele przetwarzania. Można tu wymienić systemy STRAND, PVM, P4, ANSA, SR, Emerald, PCN, czy środowiska zgodne ze standardem CORBA. Nierealne jest również dążenie do uniformizacji środowisk sprzętowych i programowych różnych organizacji czy pojedynczych użytkowników, których systemy informatyczne prędzej, czy później zaczną współpracować. Stąd wniosek, że istniejące oraz przyszłe systemy i aplikacje użytkowników są niejako ,,skazane" na współpracę. Uważamy, że przyszłe zaawansowane systemy informatyczne będą się składać ze współpracujących ze sobą modułów napisanych i uruchomionych w różnych językach i środowiskach. Prototypy takich systemów już powstały w ramach projektów badawczych [6, 7] i będziemy je tu dalej nazywać *systemami heterogenicznymi programowo*. Ich nieuniknioną właściwością jest duża złożoność, zwłaszcza na wstępnym etapie badań nad ich konstrukcją i zastosowaniem. Stąd wynika konieczność budowy odpowiednich systemów śledzenia i wizualizacji takich aplikacji. Niniejszy artykuł omawia prototypowy system tego typu, służący do monitorowania i wizualizacji heterogenicznych programowo, obiektowych aplikacji i środowisk rozproszonych. System ten, zwany MODIMOS (Managed Object-based Distributed Monitoring System), jest aktualnie realizowany w Katedrze Informatyki AGH.

Większość istniejących narzędzi monitorowania i wizualizacji nie nadaje się do środowisk obiektowych [16, 12]. Inne są najczęściej ograniczone do pojedynczego środowiska programowania [13] lub w ogóle nie działają w systemach równoległych i rozproszonych [14, 11]. Poza tym nie przewidują one współpracy komponentów wewnątrz systemu heterogenicznego programowo, a ich architektura ma często charakter zamknięty, tzn. nie są przystosowane do łatwej rozbudowy, poszerzania o nowe środowiska monitorowane. MODIMOS spełnia te wymagania. Jego użytkownik może obserwować strukturę obiektowej aplikacji rozproszonej, jej fizyczne odwzorowanie na komputery w sieci oraz zachowanie i dynamikę jej komponentów. Dzięki temu system może służyć do wykrywania nierównego rozdziału lub przeciążenia zasobów i ułatwiać ich wyważanie i dystrybucję. MODIMOS jest systemem otwartym, łatwym do rozbudowy, tzn. środowiska spełniające określone wymagania mogą być stosunkowo łatwo zaadaptowane do współpracy z systemem. W ten sposób MODIMOS może poszerzać zbiór różnych środowisk, których komponenty jest w stanie monitorować, a trzeba pamiętać, że środowisk tych jest już sporo i wciąż powstają nowe. Otwarta koncepcja systemu jest możliwa dzięki jego ustrukturalizowanej, wielowarstwowej architekturze. Definiuje ona podstawowe interfejsy oraz protokoły komunikacji, sprzęgające poszczególne warstwy ze sobą. Dotyczy to zarówno instrumentacji monitorowanego kodu, jak i mechanizmu przechwytywania i interpretacji zdarzeń, mechanizmów i technik współdziałania oraz stosowanych baz danych. Otwartość systemu jest możliwa również dzięki zastosowaniu najnowszych technik inżynierii oprogramowania, takich jak ramy [4] (ang. *frameworks*) i wzorce projektowe [5] (ang. *design patterns*).

Kluczem do efektywnego śledzenia i wizualizacji dużych i złożonych systemów, jakimi niewątpliwie są rozproszone środowiska programowo heterogeniczne, jest nawigacja zorientowana na problem, umożliwiająca użytkownikowi takie zarządzanie strumieniem informacji, by mógł ograniczyć ją do minimum niezbędnego dla rozwiązania aktualnego problemu. Stąd nacisk położony w systemie MODIMOS na mechanizmy selekcji istotnej i filtrowania nieistotnej informacji. Mechanizmy te są (lub będą) zaimplementowane na kilku poziomach systemu. MODIMOS pracuje w dwu trybach: *off-line*, umożliwiającym analizę pliku ze śladem działania aplikacji, oraz *on-line*, kiedy system nasłuchuje zdarzeń z sieci i dzięki temu praca monitorowanej aplikacji analizowana jest na bieżąco.

Niniejszy artykuł jest podzielony następująco: w sekcji 2 opisana jest architektura systemu. Sekcja 3 omawia Uniwersalny Model Obliczeniowy (UMO), przyjęty dla potrzeb projektu. W sekcji 4 przybliżone są mechanizmy współpracy różnych komponentów i środowisk programowych. Sekcja 5 skupia się na metodach wizualizacji. Artykuł kończy podsumowanie.



Rys. 1. Wielowarstwowa architektura systemu MODIMOS
Fig. 1. MODIMOS multi-layer architecture

2. Architektura systemu

Jak już wspominano, MODIMOS umożliwia jednoczesną i jednolitą pod względem logicznym obserwację kilku środowisk programistycznych (lub aplikacji zbudowanej z komponentów działających w tych środowiskach). By to osiągnąć, zaprojektowano wielowarstwową architekturę pokazaną na rysunku .

Warstwę Środowisk tworzą odpowiednio zaadaptowane obiektowe środowiska programowania rozproszonego. Praktycznie każde środowisko tego typu może być zaadaptowane na potrzeby systemu MODIMOS pod warunkiem, że dostarcza podstawowe mechanizmy komunikacji ze światem zewnętrznym (np. gniazdka – ang. *sockets*) oraz jego model obliczeniowy zawiera się w Uniwersalnym Modelu Obliczeniowym MODIMOS (patrz sekcja 3). Warstwa Środowisk składa się z dwóch poziomów: Poziomu Monitorowanych Aplikacji i Poziomu Lokalnych Monitorów. Poziom pierwszy tworzą obiekty aplikacji użytkownika, których kod źródłowy został zinstrumentowany przez odpowiednie preprocesory. Instrumentacja polega głównie na dodaniu tzw. *funkcji zawiadamiających*. Funkcje te zbierają informacje o zdarzeniach, które miały miejsce w śledzonej aplikacji i zawiadamiają o ich zajściu lokalne monitory. Monitory te to specjalne obiekty środowiska, w którym działają, mające dwa podstawowe zadania: filtrację strumienia zdarzeń wg przyjętych reguł oraz przesłanie wybranych zdarzeń przez Warstwę Współdziałania do Monitora Globalnego.

Środowiska mogą posiadać jeden lub więcej lokalnych monitorów, w zależności od swych właściwości i rozmiaru aplikacji. Ograniczanie strumienia danych wejściowych jest możliwe dzięki zastosowaniu odpowiednich reguł filtracji na obu poziomach Warstwy Środowisk. Zespół reguł tworzy *strategię* filtracji. Na Poziomie Monitorowanych Aplikacji użytkownik definiuje reguły za pomocą odpowiednich przełączników preprocesorów, pozwalających np. pominąć generację zdarzeń określonego typu lub zdarzeń dotyczących określonych obiektów. Z kolei na Poziomie Lokalnych Monitorów możliwe jest ustawianie podobnych reguł przy uruchamianiu danego monitora lub w trakcie pracy za pomocą specjalnego interfejsu zarządzania. Lokalne monitory mogą oczywiście spełniać wiele dodatkowych funkcji pomocniczych, jak kompresja danych czy translacja formatów kodowania, w zależności od potrzeb użytkownika i właściwości danego środowiska. W środowiskach usług zamkniętych (jak np. SR [2]) lokalne monitory są skonsolidowane z zasadniczą aplikacją, powstają i są niszczone wraz z nią. Z kolei w środowiskach otwartych, jak np. ANSA [1] czy CORBA [3], lokalne monitory są publicznie dostępnymi serwerami, uruchamianymi niezależnie na żądanie użytkownika.

Kolejna warstwa architektury systemu MODIMOS to Warstwa Współdziałania. Ma ona na celu zapewnienie uniwersalnej i ogólnej platformy współdziałania komponentów systemu heterogenicznego programowo, jakim jest sam MODIMOS. Warstwa ta, dzięki rozwiązaniom strukturalnym (patrz sekcja 4) i odpowiedniemu protokołowi komunikacyjnemu [9] umożliwia współpracę monitorów lokalnych różnych środowisk z Monitorem Globalnym (GM). GM przechwytuje zdarzenia nadchodzące z całego systemu monitorowanego, zbiera informację o śledzonych jednostkach i ich stanie w specjalnej bazie danych, zw. Modelem Wewnętrznym. Model ten odzwierciedla chwilową strukturę i stan tej części jednostek (obiektów) aplikacji, które wg działającej aktualnie strategii nie są pominięte (odfiltrowane). Monitor Globalny współpracuje ze znajdującym się w Warstwie Prezentacji Graficznym Interfejsem Użytkownika (GUI). Interfejs ten służy do graficznej reprezentacji struktury i stanu monitorowanej aplikacji oraz dostarcza użytkownikowi metod dodatkowych metod selekcji napływającej informacji.

3. Uniwersalny Model Obliczeniowy

System monitorowania danego środowiska programowego powinien być oparty na jego modelu obliczeniowym, tzn. śledzenie i wizualizacja powinny dotyczyć jednostek tego modelu. Tymczasem MODIMOS jest z założenia przeznaczony do zastosowania dla potrzeb wielu, często znacznie różniących się, obiektowych środowisk programowania rozproszonego. Różnice między nimi dotyczą również modeli obliczeniowych. Stąd wynikła potrzeba opracowania uniwersalnego modelu, na którym system MODIMOS mógłby być oparty. W tym celu przeanalizowano modele obliczeniowe najważniejszych dostępnych środowisk obiektowych i wyprowadzono wniosek, że ów uniwersalny, abstrakcyjny model powinien być nadzbiorem (unią) istniejących modeli. Takie podejście gwarantuje, że wszystkie możliwe w praktyce poziomy abstrakcji będą obecne w modelu uniwersalnym. Model ten nazwano Uniwersalnym Modelem Obliczeniowym (UMO), choć należy pamiętać, iż dotyczy on jedynie obiektowych środowisk rozproszonych.

Aplikacja w obiektowym środowisku programowym składa się najczęściej, na poziomie logicznym, z pewnej grupy niezależnych, komunikujących się ze sobą obiektów. Uniwersalny Model Obliczeniowy definiuje jednak więcej poziomów abstrakcji, a mianowicie: środowisko, aplikację, kontener, obiekt, interfejs, metodę i wywołanie. Odwzorowanie tych abstrakcyjnych jednostek w konkretne pojęcia wybranej grupy środowisk pokazuje tabela . By dane środowisko mogło być monitorowane w systemie MODIMOS, jego model obliczeniowy powinien zawierać się w UMO.

Tabela 1.

Przykładowe odwzorowanie UMO

Jednostka abstrakcyjna	**Środowisko**		
	ANSA	SR	Orbix
aplikacja	—	program	—
kontener	kapsuła	maszyna wirtualna	proces
obiekt	obiekt	zasób, globalium	obiekt
interfejs	interfejs	specyfikacja	interface
metoda	operacja	operacja	metoda
wywołanie	wywołanie	wywołanie	wywołanie

Poniżej przytoczono krótki opis poszczególnych jednostek abstrakcyjnych.

Środowisko. Przez środowisko jest rozumiany działający system wykonawczy (ang. *runtime system*), umożliwiający uruchomienie obiektów (zasobów) definiowanych przez użytkownika.

Aplikacja. Pojęcie aplikacji nie jest trudne do zdefiniowania w środowiskach zamkniętych, jak np. SR. Tworzą ją tam wszystkie obiekty i zasoby zdefiniowane przez użytkownika w danym programie, najczęściej wraz z dedykowaną instancją środowiska dla ich wykonania. Inaczej ma się sprawa ze środowiskami otwartymi, jak np. ANSA czy

CORBA, gdzie działają ogólnie dostępne serwery usług, uruchamiane i wykorzystywane przez różnych użytkowników. Nie da się tam zakreślić granic między aplikacjami w sposób arbitralny czy automatyczny. Jedynie dany użytkownik jest w stanie zdefiniować zasoby, wchodzące w skład jego aplikacji.

Kontener. Kontenery to obszary pamięci dzielone przez obiekty w nich rezydujące. Zazwyczaj kontenery są implementowane przez procesy systemu operacyjnego.

Obiekt. Definicja obiektu jest tradycyjna i powszechnie znana. Przypomnimy tylko, że w niektórych środowiskach (np. ANSA) obiekt może mieć więcej niż jeden interfejs.

Interfejs. Interfejs definiuje metody i atrybuty, których obiekt dostarcza dla użytku innych obiektów.

Metoda. Metoda to operacja wchodząca w skład interfejsu obiektu. MODIMOS rozróżnia i pozwala śledzić osobno instancje metod każdego obiektu, czyli w istocie wątki wykonania związane z daną operacją.

Wywołanie. Wywołanie jest abstrakcyjnym pojęciem reprezentującym stan, w którym obiekt–klient wywołuje usługę (metodę) z interfejsu obiektu–serwera. Sumaryczna liczba wywołań powinna być równa liczbie instancji metod w aplikacji. Takie podejście umożliwia spójne i symetryczne śledzenie i wizualizację komunikacji w obiektowym systemie rozproszonym.

Z każdym elementem Uniwersalnego Modelu Obliczeniowego związna jest para komplementarnych zdarzeń: utworzenie i zniszczenie danego elementu. Zdarzenia te są raportowane przez funkcje zawiadamiające do lokalnych monitorów i stamtąd dalej do Monitora Globalnego. Stąd z każdą jednostką UMO związane są dwie funkcje notyfikujące. Zbiór funkcji zawiadamiających jest w sposób uniwersalny opisany interfejsem monitorowania w języku IDL (*Interface Definition Language*) [3], co umożliwia łatwe przenoszenie do różnych języków i środowisk. Interfejs ten definiuje zarówno interfejsy monitorujące wszystkich lokalnych monitorów, jak i Monitora Globalnego.

Aby rozszerzyć system MODIMOS o nowe środowisko należy:

- przeanalizować jego model obliczeniowy i znaleźć jego odwzorowanie do UMO,
- dokonać translacji interfejsu monitorowania (funkcji zawiadamiających) na język programowania danego środowiska,
- napisać odpowiedni preprocesor instrumentujący kod źródłowy aplikacji w danym środowisku
- zaimplementować monitor(y) lokalny(e) na podstawie interfejsu monitorowania

- zaimplementować fragment podsystemu komunikacji w Warstwie Współdziałania

Osobny problem w systemach heterogenicznych programowo stanowi unikalna identyfikacja obiektów. Każde środowisko posiada własny, niekompatybilny z innymi, system identyfikacji. Tymczasem w aplikacji heterogenicznej programowo, a zwłaszcza w systemie monitorowania, istnieje potrzeba istnienia globalnie unikalnych identyfikatorów. Można rozwiązać ten problem, wykorzystując oryginalne identyfikatory z poszczególnych środowisk i trakując je jako ciągi bajtów, ew. uzupełnione krótkimi przedrostkami, nadającymi im globalną unikalność. Takie identyfikatory „naturalne" są jednak długie i stosowanie ich jako kluczy przeszukiwań w wielu typach baz danych powodowałoby nieefektywność operacji wyszukania i zapisu. Stąd w systemie MODIMOS zastosowano „sztuczne" identyfikatory o strukturze hierarchicznej, tworzone przez kod instrumentujący specjalnie na potrzeby systemu. Identyfikator hierarchiczny składa się z komponentów, z których każdy odpowiada pewnemu poziomowi logicznemu UMO i zawiera liczbę instancji elementów danego poziomu w elemencie nadrzędnym. Taki typ identyfikatora, kodowany binarnie lub znakowo, znacznie usprawnia operacje na niektórych typach baz danych, stosowanych w projekcie (patrz sekcja 5).

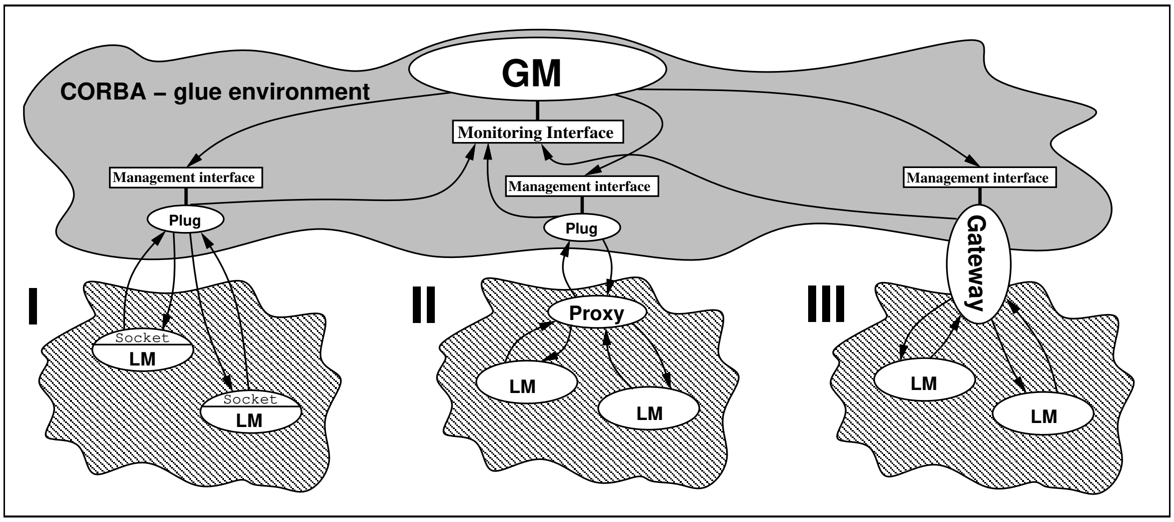
4. Mechanizmy współdziałania

Monitorowane środowiska z reguły charakteryzują się odmiennymi rozwiązaniami architektonicznymi, a komunikacja między obiektami aktywnymi w danym środowisku odbywa się zgodnie ze specyficznym dla niego protokołem. Powoduje to, że nawiązanie bezpośredniej komunikacji między monitorowanymi aplikacjami a Globalnym Monitorem jest niemożliwe. Rozwiązanie tego problemu polega na wprowadzeniu do architektury systemu monitorowania specjalnej Warstwy Współdziałania, której zadaniem jest zapewnienie uniwersalnego mechanizmu wywołania odległych operacji pomiędzy lokalnymi monitorami, aktywnymi w poszczególnych środowiskach, a GM. Podsystem komunikacji Warstwy Współdziałania może być zrealizowany jedną z dwu przedstawionych poniżej metod.

Budowa od podstaw mechanizmu komunikacyjnego, dedykowanego dla Warstwy Współdziałania, opartego na jednym z prymitywnych interfejsów komunikacyjnych, takich jak np. mechanizm gniazdek (ang. *sockets*).

Zastosowanie łączącego środowiska rozproszonego (ang. *glue environment*), takiego jak Orbix [10], ANSA [1] lub ToolTalk, dostarczającego, oprócz mechanizmów komunikacyjnych, szereg dodatkowych usług, takich jak: lokalizacja serwerów, kojarzenie żądań klientów z ofertami serwerów (ang. *trading*), rozsyłanie wywołań (ang. *dispatching*), itd. Koncepcję wykorzystania środowiska łączącego przedstawiono na rys. 2.

Największą zaletą pierwszego z tych rozwiązań jest jego prostota, która pociąga za



Rys. 2. Warianty realizacji Warstwy Współdziałania
Fig. 2. Variants of Interoperability Layer architecture

sobą dużą efektywność – transmisja danych między lokalnymi monitorami a Globalnym Monitorem jest bezpośrednia i możliwe jest wysłanie dużej ilości komunikatów w jednostce czasu. Takie rozwiązanie utrudnia jednak konfigurowanie Warstwy Współdziałania, a wszystkie potrzebne do tego mechanizmy muszą być zaprojektowane i wykonane od podstaw.

W drugim rozwiązaniu komunikaty przesyłane są za pomocą mechanizmu wywołań odległych operacji, dostarczanego przez środowisko łączące. Zwalnia to programistę z konieczności pracochłonnej implementacji własnej infrastruktury komunikacyjnej (modułów odbierających i wysyłających, buforujących i interpretujących komunikaty). Możliwe jest także wykorzystanie mechanizmów konfiguracji, dostępnych w środowisku łączącym. Pozwala to na rekonfigurację Warstwy Współdziałania podczas pracy systemu. Drugie rozwiązanie zapewnia także pożądaną strukturalizację Warstwy Współdziałania. Pewną wadą jest natomiast mniejsza, w stosunku do prymitywnych mechanizmów komunikacyjnych, efektywność przetwarzania. Jednakże opisywany system ma z założenia umożliwiać przede wszystkim selektywną obserwację monitorowanych środowisk, a więc ilość przesyłanej informacji jest zazwyczaj znacznie zredukowana przez użytkownika.

Łączące środowisko rozproszone musi zostać zintegrowane ze środowiskami monitorowanymi. Najlepszym rozwiązaniem byłaby konstrukcja obiektu-bramy [8] (ang. *gateway*), który byłby aktywny jednocześnie w środowisku łączącym i w danym środowisku monitorowanym (wariant III na rys. 2). Oznacza to, że taki obiekt mógłby wysyłać i odbierać wywołania z obu tych środowisk. Niestety, wiele środowisk nie dopuszcza takiej możliwości.

Alternatywne rozwiązanie polega na zaprojektowaniu i implementacji obiektu–wtyczki (ang. *plug*), który jest aktywowany w środowisku łączącym i reprezentuje w nim dane środowisko monitorowane. Wtyczka odbiera od lokalnych monitorów w danym środowisku polecenia wywołań odległych operacji i dokonuje ich konwersji do formatu środowiska łączącego. Środowisko łączące umożliwia komunikację wtyczek z GM, pozwala na lokalizację GM i przekazuje do niego wywołania operacji.

Zostały wyróżnione dwa możliwe warianty konstrukcji wtyczek i ich komunikacji z lokalnymi monitorami. Wybór jednego z nich zależy od cech, jakie posiada dane środowisko monitorowane.

Rozwiązanie z użyciem gniazdek. Komunikacja między lokalnymi monitorami a wtyczką realizowana jest w oparciu o mechanizm gniazdek (wariant I na rys. 2). Rozwiązanie to powinno być stosowane wtedy, gdy dane środowisko nie oferuje wątków ani innego mechanizmu pozwalającego na implementację komunikacji asynchronicznej. Wówczas musi być stosowany mechanizm zdarzeniowy oparty na sygnałach. Przykładem środowisk wymagających stosowania powyższej metody są Strand i PCN.

Rozwiązanie z obiektem-reprezentantem (ang. *proxy*). W komunikacji między lokalnymi monitorami a wtyczką pośredniczy specjalny obiekt aktywny w danym środowisku monitorowanym, reprezentujący w nim GM (wariant II na rys. 2). Monitory lokalne komunikują się z reprezentantem za pomocą mechanizmu wywołania właściwego dla danego środowiska, natomiast komunikacja reprezentanta z wtyczką odbywa się za pomocą mechanizmu gniazdek.

Warstwa Współdziałania systemu MODIMOS została zrealizowana przy wykorzystaniu środowiska łączącego, którego rolę pełni system Orbix. Dotychczas do warstwy współdziałania dołączono dwa środowiska monitorowane: ANSA i SR. W celu ich integracji ze środowiskiem łączącym zastosowano mechanizm wtyczek i reprezentantów (wariant II na rys. 2).

5. Globalny Monitor i Warstwa Prezentacji

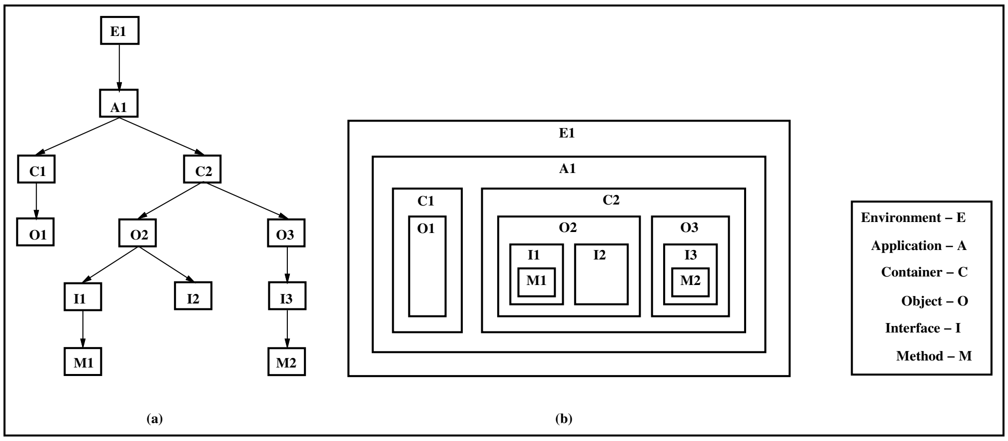
Warstwa prezentacji systemu monitorowania realizuje dwie niezależne funkcje: dostarcza interfejs pozwalający na zarządzanie systemem oraz jest odpowiedzialna za wizualizację zdarzeń zachodzących w monitorowanych środowiskach. Aplikacje działające w poszczególnych środowiskach składają sie z elementów tworzących strukturę hierarchiczną, którą można opisać zarówno modelem obliczeniowym danego środowiska, jak i Uniwersalnym Modelem Obliczeniowym wspólnym dla wszystkich środowisk. Informacje o aktualnym stanie monitorowanych elementów oraz o wiążących je zależnościach są gromadzone w Globalnym Monitorze i zapisywane w bazie danych, zwanej Modelem Wewnętrznym (MW). Każdy monitorowany element jest reprezentowany przez pewien obiekt Modelu Wewnętrznego. W dowolnym momencie pracy systemu logiczna struktura powiązań obiektów Modelu Wewnętrznego odpowiada strukturze monitorowanych elementów.

Dla implementacji MW kluczowe znaczenie ma wybór odpowiedniej struktury danych zapewniającej dużą efektywność operacji wyszukiwania, wstawiania i usuwania obiektów reprezentujących monitorowane elementy. Jednocześnie należy zauważyć, że wybór struktury danych zależy silnie od założeń, jakie zostały narzucone na identyfikatory monitorowanych elementów. Na przykład, jeśli stosowane są identyfikatory hierarchiczne, to powinna zostać wybrana struktura danych wykorzystująca ten fakt w celu zwiększenia efektywności wymienionych wyżej operacji.

Z tego powodu, a także w celu umożliwienia łatwego testowania różnych struktur danych, GM został skonstruowany w sposób pozwalający na łatwą wymianę struktury danych wykorzystywanej w MW. Wykonano szereg imlementacji MW opartych na różnych strukturach, takich jak tablice rozproszone, B-drzewa oraz struktury drzewiasto-listowe. Wykorzystano przy tym gotowe komponenty dostępne w bibliotekach obiektowych, takich jak Tools.h++ i Booch Components oraz struktury zaimplementowane własnoręcznie. Obecnie przeprowadzane są pomiary efektywności przetwarzania zrealizowanych implementacji, mające na celu wybór najlepszej struktury danych dla poszczególnych rodzajów identyfikatorów.

Informacje gromadzone w MW są na bieżąco przetwarzane i wyświetlane przez interfejs graficzny. Powstawanie nowych obiektów w MW, niszczenie oraz zmiany stanu obiektów już istniejących, powodowane przez odpowiednie zmiany w monitorowanych aplikacjach, są wizualizowane przez interfejs graficzny. Ponieważ monitorowane elementy (oraz obiekty MW) tworzą strukturę hierarchiczną (drzewiastą), zadaniem o kluczowym znaczeniu jest opracowanie efektywnej metody wizualizacji drzew. Ponadto, ze względu na stosunkowo dużą liczbę obiektów gromadzonych w MW, konieczne jest dostarczenie użytkownikowi wygodnych i efektywnych mechanizmów wyboru interesujących go fragmentów struktury monitorowanych elementów.

Wizualizowane elementy są reprezentowane na ekranie w postaci figur geometrycznych. Atrybuty każdej figury (kształt, kolor) zależą od rodzaju wizualizowanego elementu (tj. jego poziomu w UMO). Wykonanie określonej akcji na figurze powoduje wyświetlenie szczegółowych informacji o elemencie (np. o jego fizycznym położeniu). Wizualizacja hierarchicznej struktury elementów może być zrealizowana przy użyciu grafiki trójwymiarowej (np. drzew stożkowych [15]) albo w przestrzeni dwuwymiarowej. W pierwszej wersji systemu zastosowano reprezentację dwuwymiarową. Wyróżniono przy tym dwa rodzaje tej reprezentacji: drzewiastą (rys. 3 a) i skrzynkową (rys. 3 b), różniące się między sobą sposobem oznaczania powiązań między węzłami. W reprezentacji drzewiastej, która jest stosowana w obecnej wersji systemu, relacja ojciec-syn jest reprezentowana przez linie, łączące odpowiednie figury. W reprezentacji skrzynkowej figury reprezentujące węzły dzieci są umieszczane wewnątrz figur reprezentujących węzły ojców.



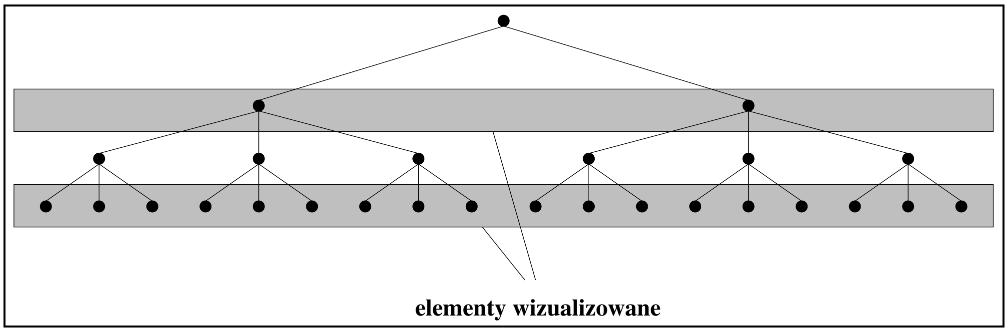
Rys. 3. Dwie metody dwuwymiarowej reprezentacji struktur hierarchicznych: a) drzewiasta i b) skrzynkowa

Fig. 3. Two methods of tree's presentation: a) tree-like and b) box-like

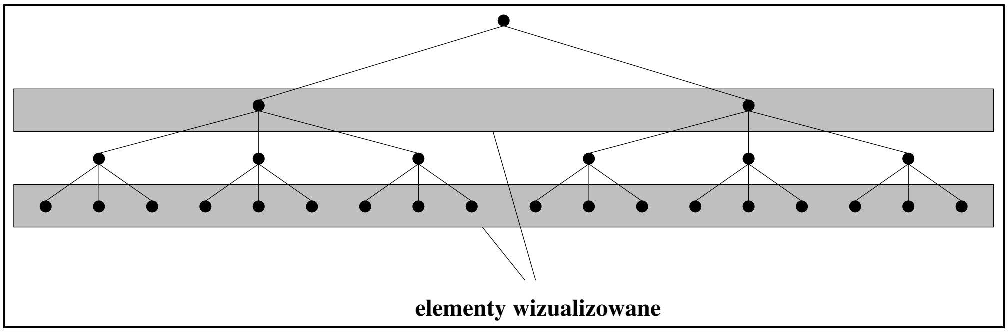
Pomimo stosowania mechanizmów filtracji w niższych warstwach systemu MODIMOS liczba elementów reprezentowanych w MW może być na tyle duża, że ich efektywna wizualizacja będzie niepożądana lub nawet niemożliwa. Dlatego warstwa prezentacji dostarcza machanizmy selekcji, pozwalające na redukcję rozmiarów drzewa wyświetlanego na ekranie. Dwie podstawowe metody selekcji to selekcja pozioma i pionowa. Selekcja pozioma (rys. 4) polega na eliminacji wybranych poziomów aktualnie wyświetlanego drzewa (tj. usunięcie z ekranu figur reprezentujących elementy określonego poziomu UMO, np. interfejsów albo kontenerów). Selekcja pionowa (rys. 5) umożliwia natomiast usunięcie wybranych poddrzew aktualnie wyświetlanego drzewa. Pozwala więc na usunięcie z ekranu informacji o nieinteresujących dla obserwatora elementach i ich zawartości (np. figury reprezentującej nieinteresujący użytkownika kontener wraz ze wszystkimi jego obiektami).

Oprócz mechanizmów wyboru elementów warstwa prezentacji pozwala na maskowanie wizualiowanych zdarzeń. Możliwe opcje o praktycznym znaczeniu to np.: ignorowanie wszystkich zdarzeń (tj. zamrożenie obrazu aktualnie wyświetlanego w oknie), ignorowanie zdarzeń dotyczących tworzenia nowych elementów (tj. ograniczenie obserwacji do elementów aktualnie wizualizowanych).

Opisane mechanizmy selekcji i filtracji mogą okazać się niewystarczające w sytuacji, gdy użytkownik chce obserwować jednocześnie różne części monitorowanej struktury elementów. Dlatego interfejs graficzny pozwala na jednoczesną obserwacje dowolnych fragmentów struktury MW w różnych oknach. W każdym z nich może być wyświetlane dowolne poddrzewo (lub jego fragment) hierarchii reprezentowanej w MW. Dany element



Rys. 4. Selekcja pozioma
Fig. 4. Horizontal selection



Rys. 5. Selekcja pionowa
Fig. 5. Vertical selection

może być jednocześnie reprezentowany w wielu różnych oknach. Zmiana korzenia aktualnie wyświetlanego podrzewa polega na wyborze jednej z figur widocznych w danym oknie, która staje się nowym korzeniem.

6. Posumowanie

W artykule przybliżono projekt MODIMOS – prototypowy system monitorowania i wizualizacji heterogenicznych programowo, obiektowych aplikacji rozproszonych. Zrealizowano już zasadniczy zrąb projektu: Warstwę Środowisk, Współdziałania i Monitor Globalny wraz z kilkoma wersjami Modelu Wewnętrznego. Obecnie trwają prace nad implementacją interfejsu graficznego na podstawie obiektowej biblioteki Fresco i środowiska graficznego X-Window. W rozwiązaniu tym każdy element Modelu Wewnętrznego będzie mógł być reprezentowany przez wiele obiektów przedstawiających figury lub okna. W początkowej wersji systemu GM oraz interfejs graficzny będą zintegrowane w jednym procesie. W przyszłości natomiast zostanie wykorzystane jedno z rozproszonych środowisk w celu oddzielenia tych modułów. Wówczas obiekty graficzne będą jednocześnie obiektami środowiska rozproszonego. Pozwoli to na tworzenie wielu instancji interfejsu graficznego fizycznie oddalonych od Monitora Globalnego i da możliwość jednoczesnej obserwacji zmian w monitorowanej aplikacji przez wielu użytkowników.

LITERATURA

[1] ANSAware 4.0 – Application Programmer's Manual, APM Ltd. Cambridge, 1992.

[2] Andrews G.R., at el.: An overview of the SR language and implementation, *ACM Trans. on Progr. Languages and Systems*, 10(1):51–86, January 1988.

[3] Draft Common Object Request Broker Architecture Revision 1.1, OMG Report 91-12-1, OMG Inc., 1991.

[4] Campbell R., Islam N.: A Technique for Documenting the Framework of an Object-Oriented System, *Proc. IWOOS'92*, IEEE Computer Society Press, Los Alamitos, CA, September 1992.

[5] Gamma E., Helm R., Johnson R., Vlissides J.: *Design Patterns: Elements of Object-Oriented Software Architecture*, Addison-Wesley, 1994.

[6] Nierstrasz O., Gibbs S., and Tsichritzis D.: Component-Oriented Software Development, *Comm. of the ACM*, 35(9):160–165, September 1992.

[7] Udell J., Componentware, *Byte* vol. 19 no.5, pp. 46-56, May 1994.

[8] Uszok A., Czajkowski G., Zieliński K.: Interoperability Gateway Construction for Object-Oriented Distributed Systems, *Proc. of the 6th Nordic Workshop on Programming Environment Research*, Lund, Sweden, TR of the Lund University, June 1994.

[9] Czajkowski G., Uszok A., Zieliński K.: Distributed Declarative Systems as Parts of Cooperating Software Environments, *Proc. of the ICLP'94 Post-Conference Workshop on Integration of Declarative Paradigms*, Santa Margerita, Italy, June 1994.

[10] The Orbix Architecture, IONA Technologies Ltd., 1993.

[11] Jerding D.F. and Stasko J.T.: Using Visualization to Foster Object-Oriented Program Understanding, Georgia Institute of Technology, TR GIT-GVU-94-33.

[12] Reed D.A., Aydt R.A., Noe R.J., Roth P.C., Shields K.A., Schwartz B., Tavera L.F.: Scalable Performance Analysis: The Pablo Performance Analysis Environment, In *Proc. of the Scalable Parallel Libraries Conference*, IEEE Computer Society Press, 1993.

[13] Jamrozik H., Roisin C., Santana M.: A Graphical Debugger for Object-Oriented Distributed Programs, *Proc. TOOLS'91*, pp. 117-128, July 1991.

[14] Pauw W.De, Helm R., Kimelman D. Vlissides J.: Visualizing the Behavior of Object-Oriented Systems, *Proc. OOPSLA '93*, ACM Press pp. 326-337, SIGPLAN, Washington D.C, October 1993.

[15] Vion-Dury J.Y., Santana M.: Virtual Images: Interactive Visualization of Distributed Object-Oriented Systems, *Proc. OOPSLA'94*, Portland USA, October 1994.

[16] Miller B.P., Clark M., Hollingsworth J., Kierstead S., Lim S., Torzewski T.: IPS-2: The Second Generation of a Parallel Program Measurement System, *IEEE Transactions on Parallel and Distributed Systems* 1, 2 (April 1990), pp. 206-217.

Recenzent: Dr hab. inż. Tadeusz Czachórski

Wpłynęło do Redakcji 21 grudnia 1995.

Abstract

The next decade will bring radical changes to the way we perform information processing, as applications composed of many cooperating distributed subsystems, exploiting different computational paradigms, become more common. This situation increases substantially demands in the area of system management, correctness analysis, understanding, debugging and performance evaluation. Managed Object-based Distributed Monitoring System (MODIMOS) is a project aimed at development of an adaptable platform for visualization of distributed applications, built of interoperating heterogeneous components. In other words, the system serves for monitoring and visualization of applications built of many cooperating components, written in different computing environments. MODIMOS is expandable, allowing to add new monitored environments. It employs various management mechanisms to cope with gathered information size and complexity. The expandability of MODIMOS architecture has been proved by successful integration of ANSA, SR and Orbix within the Interoperability Layer [8]. This provided the major feedback for the applied technical solutions and proved correctness of this layer's concept. This paper describes also the multi-layer architecture of the system (Fig. 1) and the possible visualization schemes (Fig. 3).